Final Report

Contract No. F49620-C-91-0098

Approved for public release,
distribution unlimited

Richard Tolimieri

Aware Inc.
One Memorial Drive,
Cambridge, MA 02145.

December 6, 1994

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		FINAL/30 SEP 91 TO 31 DEC 94

4. TITLE AND SUBTITLE
EFFICIENT & FLEXIBLE ALGORITHMS FOR DIGITAL SIGNAL PROCESSING ON MULTIPLE INDEPENDENT NODE PARALLEL COMPUTERS

5. FUNDING NUMBERS
8099/03
F49620-91-C-0098

6. AUTHOR(S)
DR. TOLIMIERI

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AWARE INC
ONE MEMORIAL DRIVE
CAMBRIDGE, MA 02145

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 95-0429

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-91-C-0098

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
During the last 3 years we have developed a mathematical theory of algorithms and implementation strategies for DSP computations on RISC and DSP chips and parallel architectures ranging from scalable multinode boards to massively parallel multinode computers as typified by the Intel's touchstone systems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

Table of Contents

Appendix E: Group Invariant Fourier Transform Algorithms

1 Overview

During the last 3 years we have developed a mathematical theory of algorithms and implementation strategies for DSP computations on RISC and DSP chips and parallel architectures ranging from scalable multinode boards to massively parallel multinode computers as typified by the Intel's Touchstone systems.

Recently, our work has centered around implementation of the DFT, convolution and wavelet multirate filter systems on distributed parallel computing platforms, and embedding of the routines in various applications in collaboration with several government laboratories, commercial institutions and university research groups.

The general goal of this effort is to establish tools which apply concurrently to software and hardware and create

- a technology base for developing optimal software, extending the life span of software by appropriately targeting suitable hardware.
- procedures for cost effective system design for special purpose architectures which can be expected to efficiently implement a whole class of similar algorithms of interest.
- immediate utilization of new hardware advances at minimal time and cost in software development.

The director of the group is Richard Tolimieri who is partially supported by the contract. The contract also supports Myoung An full time, Chao Lu of Towson University as a consultant, and three graduate students two of whom have received PhD during this period.

One feature of our approach is that algorithms are modeled in algebraic terms permitting software to be optimized by algebraic manipulations as oppose to more time-consuming programming manipulations. This algebra identifies and operates on fundamental computational and communication primitives which concurrently model software and machine parameters and establishes interactive programming tools in the form of transformation rules for selecting highly optimized code for a target architecture.

We have developed a theory of algorithms for DSP computations based on finite abelian group theory that divorces the problem of algorithm and system design from the particulars of implementation and application and has led to the development of new algorithms which present radically different communication paths and data structures for subcomputations. This is especially important in multidimensional processing which incorporates more degrees of freedom for system and algorithm design but involves data sizes that challenge hardware memory resources, I/O and interprocessor communication bandwidth. In this framework, new algorithms have been

designed for incorporating special data characteristics (real, hermitian, space group symmetric) and for embedding code in applications highlighting special local data characteristics. Typically such applications involve iteration of distinct computations where standard algorithms result in a mismatch between input and output data structures of successive stages.

These tools have and will significantly impact computations in such diverse application areas as image processing, x-ray crystallography, communications, computational fluid dynamics and computational electromagnetics.

We have applied our results summarized below in collaboration with government agencies, universities and commercial institutions.

1.1 Applied Results

goals:	accomplishments:
• Develop a theory for data partition and migration on shared and distributed memory multiprocessors.	• Formulation of data partitioning and migration schemes in terms of tensor product algebra.
• Implementation of the theory developed for data partitioning and migration in parallel solutions for applications.	• Implementation of routines to interface various data partitioning in distributed computing systems for general numerical procedures involving sequences of computations requiring intermediate data redistribution. • Implementation of matrix multiplication using the theory to change the data flow from existing matrix multiplication algorithms. • Interface multidimensional FFT for the wavelet-Galerkin and capacitance matrix methods for the solutions of Euler and Navier-Stokes equations.
• Improve the efficiency of Intel's multidimensional FFT library.	•Interleaved communication and computation in the 3D FFT, along with the use of efficient vectorized assembly FFT codes improves the 3D FFT code up to 50 %. • Tensor product formulation of the 2D FFT allows for maximizing the degree of concurrency between computations of row 1D FFTs and global transposition to result in up to 40 % faster codes.

goals:	accomplishments:
• Create scalable 1D and 2D power of 2 and composite transform size parallel DFT library using reduced interprocessor communication variants of MD Cooley-Tukey and Good-Thomas algorithms.	• A family of M-D implementations improving performance up to 200% over powers of 2 Intel 2D and 3D code.
• Create a scalable 1D and 2D composite transform size parallel DFT library on the Intel IPSC/860 based on standard row-column method.	•a scalable library of composite size 1D and 2D parallel DFT implementations with CPU compatible with $n \log n$ criteria.
• Create a scalable 2D and 3D library of parallel DFT codes based on the vector-radix algorithm and compare their performance with the row-column approach.	• A scalable library of 2D and 3D vector-radix implementations along with a comparison with row-column implementations and identification of cases where vector-radix outperforms row-column method.
• Create a scalable library of efficient non-powers-of-two parallel DFT codes with reduced inter-processor communication needs, using variants of the RTA algorithm.	• A family of RTA variants implementations improves the performance of the parallel DFT up to 75 % over the powers-of-two Intel 2D and 3D FFT code.
• Investigate the suitability of the parallel algorithms we proposed for other parallel multiprocessor systems (Clusters of workstations).	• Parallel RTA variants coded to run on a cluster of SUN workstations show promising speedup and scalability features.

goals:	accomplishments:
• Develop of a library of parallel symmetrized DFT codes.	• Derivation of a novel symmetrized DFT algorithm based on group theoretic concepts, implementable on multi-processor machines, with a wide range of applications in crystallography and signal processing.
• Investigate integer and rationally oversampled Weyl-Heisenberg coefficient computation in a distributed memory multiprocessor environment.	• A library of real time implementation of integer and rationally oversampled Weyl-Heisenberg coefficient computation on single i860 processor and on 4- and 8-node computing systems.
• Documentation of employed methods of design and implementation of parallel algorithms in the most widely available form for the purpose of immediate availability by the public.	• In addition to publication of *Mathematics of Multidimensional Fourier Transform Algorithms*, Springer-Verlag textbook, several papers to journals have been submitted and presentations were given at conferences.
• Porting of Touchstone parallel codes to other parallel architectures as a test of portability of our methods.	• The parallel FFT codes have been successfully ported to the IBM SP2 multiprocessor system of the NAS NASA Research Center in less than a day.

These applied results have led to the following technology transfers.

1.2 Technology transfer

- David Grimm, Honeywell, Inc., 813 539 4213

  Embeddable Multiprocessor Systems

  We have ported scalable, multiprocessor, multidimensional FFT routine for variable size PARAGON systems. Honeywell has agree to act as β-site for the codes and the given machine environment for the codes we have developed.

- A. King, Intel Corporation, Supercomputer Systems Division, 503 531 5300.

  - i860

    For the Intel i860, we have developed a library of mixed size FFT routines, which will soon be available in the commercial market. The library is three times denser in transform sizes than existing such libraries. The non-powers-of-two sizes run at the linear time scale as the powers-of-two sizes which run competitively with assembly coded fully optimized routines in other libraries.

  - Touchstone Systems, DELTA, iPSC/860, PARAGON

    For the Intel Touchstone systems, we have implemented scalable, multiprocessor, multidimensional FFT routines optimized for each of the three systems.

- E. Prince, Reactor Division, NIST, 301 970 6230.

  X-ray crystallographic FFT routines.

  SUN, Microways's NumberSmasher860 accelerator card.

  We worked with Dr. Edward Prince of NIST to embed our crystallographic group specific mixed size FFT library. For computational methods in X-ray crystallography, mixed size FFT routines are crucial. Library was created in collaboration with Dr. Prince to address the most applicable computations for compile-time efficiency. During our collaboration, Dr. Prince has changed his computing environments three times, VAX, 486 PC and most recently added i860 accelerator card for compute intensive procedures. In each of the computing environments, our codes have significantly improved (3 - - 100 times) the runtime of the computations.

- J. Weiss, Aware, Inc., 617 577 1700.

  Computational Fluid Dynamics

2D FFT on Intel's Touchstone systems

We supplied Dr. Weiss of Aware, Inc. with data-restructuring routines for Intel's Delta machine for his parallel methods for incompressible Euler and Navier-Stokes equations for fluid dynamics in two-space. While parallelization of other computational procedures required non-traditional data structures, parallel optimized FFT routines are available only for row-column distributed data structure. Our data restructuring routines are formulated in terms of global/local stride permutations, and embeddable in row-column distributed FFT routines. In fact, we have improved the global FFT routines by 120-200%.

- C. Lund, Mercury Computer Systems, Inc., 508 256 1300.

  Mercury's MCV6

  We are working to port our parallel multidimensional FFT routines and Weyl-Heisenberg coefficient computation routines to Mercury's four-node board.

- C. Giacovazzo, Departmento Geomineralogico, Campus Univarsitario, Bari, Italia, 39 80 544 2590.

  - SUN

    We are developing optimized cubic-symmetry-specific FFT code for Dr. Giacovazzo of University of Bari, Italy.

  - i860-based multiprocessor boards.

    We are parallelizing Dr. Giacovazzo's software package for small molecule direct methods, SIR92, for an i860-based multiprocessor boards.

- A. Woo, NASA AMES Research Center, 415 604 6010.

  Computational Electromagnetics.

  Intel's PARAGON, IBM's SP2.

  We have ported mixed-size 3-dimensional parallel FFT code for Intel Paragon and IBM's multiprocessor SP2 for applications in computational electromagnetics.

- G. Tennille, NASA Langley Research Center, 804 864 5786.

  Intel's PARAGON, IBM's SP2.

  We have transferred multi-dimensional double precision FFT routines for the Intel PARAGON and in the process of transferring similar codes on IBM's SP2.

- E. Bleszynski, Rockwell International Corporation, North American Aircraft Operations, 310 647 3675.

  Adaptive Integral Method Solver of Large Scale Electromagnetics Computations.

  Intel's Paragon

  We have transferred a package of scalable mixed-radix 3-dimensional FFT routines for Paragon nodes.

- E. Holbert, Kirtland AFB, 505 846 1995.

  SUN

  We have transferred FT routines of sizes 1000 and 1024 for real data sequences optimized for the SUN.

- R. Pachter, Wright Laboratory, WPAFB, OH, 513 255 6652.

  Intel's Paragon

  Materials Science

  We have ported real/Hermitian 2-dimensional parallel FFT routines to the material science division for Paragon.

- R. Martino, Department of Computer Science and Engineering, NIH, 301 496 1111.

  Intel's iPSC/860

  Molecular Dynamics

  We have ported 3-dimensional parallel FFT routines for iPSC/860 128 node hypercube. The performance recorded by NIH of our code was 1.2Gbyte running FFT which is highest recorded to our knowledge.

- Steven Fried, Microway, Inc. 508 585 1277.

  During the last year we have actively collaborated with Microway, Inc. to produce a library for their i860 accelerator card, that is three times denser in transform sizes than existing such libraries. This library was ported to Dr. E. Prince of NIST for interface with his crystallographic procedures and resulted in a speed-up of twenty times. It will soon be commercially available through Microway, Inc. Presently, we are collaborating on producing scalable, mixed-radix, parallel FFT library for the quadputer i860 board. We have access to Microway's hardware products in these joint efforts.

- M. Tzannes, Aware, Inc., 617 577 1700

  Tele-communication project.

  - The converting fixed-point input to floating-point stage was combined with the inverse 512-point FFT to save arithmetic operations to produce optimal code on ADSP21020.
  - The Frequency-Equalizer stage was combined with the 512-point FFT to save arithmetic operations to produce optimized code on ADSP21020.
  - 512-point DCT II and IV (Discrete Cosine Transform) have been optimized for Analog Devices's ADSP-21020 chip based on FFT and will be optimized on the new ADSP21060.
  - 512-point DWMT (Discrete Wavelet Multitone Technique) modulator and demodulator based on DCT IV has been optimized on ADSP-21020, and will be optimized on the new ADSP21060.

- Loral Federal Systems Inc. — Benchmark on the IBM SP2 project.

  - QUICK_CPF.F
  - QUICK_CPF_COMPRESSED.F
  - IPF.F
  - QUICK_IPF.F

  the above 4 routines were optimized on IBM SP2 parallel systems, with about 30% improvement.

- Atlantic Aerospace — FIR filter on ISP multi-processor board based on TI TMS320C40 chip.

  - 4-point FIR filter
  - 8-point FIR filter

1.3 Publications

1. *Mathematics of Multidimensional Fourier Transform Algorithms,* R. Tolimieri, M. An and C. Lu, Springer-Verlag, New York, 1993.

2. "Efficient Multidimensional FFT Module Implementation on the Intel i860 Processor", M. An, C. Lu, S. Qian and R. Tolimieri, Proc. Inter. Conf. on Signal Processing, Applications and Technology, Sept. 28-Oct. 1, CA 1993.

3. "A Hybrid Parallel FFT Algorithm Without Interprocessor Communication", M. An, C. Lu, S. Qian and R. Tolimieri, Proc. of IEEE Inter. Conf. on ASSP, 1993.

4. "DSP algorithm design and implementation on RISC architectures," M. An, C. Lu and R. Tolimieri, presented at the first international conference on Electronics and Information Technology (ICEIT'94) held in Beijing, China.

5. "The computation of Weyl-Heisenberg coefficients for critically sampled and oversampled signals," M. An, G. Kechriotis, C. Lu and R. Tolimieri, presented at ICSPAT '94, Dalla, TX.

6. "Self-Sorting In-Place FFT Algorithm with Minimum Working Space," by Z. Qian, C. Lu, M. An and R. Tolimieri, *IEEE Transactions on Signal Precessing*, **42** 10, October 1994.

7. "A New Approach for Computing Multi-Dimensional DFTs on Parallel Machines and its Implementation on the iPSC/860 Hypercube," M. An, M. Bletsas, G. Kechriotis, E. Manolakos and R. Tolimieri, to appear in *IEEE Trans. ASSP.*, January, 1995.

8. "Group Invariant Fourier Transform Algorithms," Y. Abdelatif, M. An, N. Anupindi, G. Kechriotis, C. Lu and R. Tolimieri, to appear as a chapter in *Advances in Electronics and Electron Physisc*, Acadimic Press.

9. "Comparison of 2-D FFT Implementations on the Intel Paragon Massively Parallel Supercomputer", M. An, N. Anupindi, M. Bletsas, G. Kechriotis, C. Lu, E. S. Manolakos and R. Tolimieri, to appear at *Proc. of the International Conference on Speech, Acoustics and Signal Processing (ICASP)*, April 1994.

10. "Weyl-Heisenberg systems and the finite Zak transform," M. An, A. Brodzik, G. Kechriotis, C. Lu and R.Tolimieri, submitted for publication in *Signal Processing*, Elsevier Science Publishers, Amsterdam, The Netherlands, August 94.

11. "Group algebras and orthogonal decompositions," M. An and R. Tolimieri, submitted to *IEEE SP* for publication, June, 1994.

2 Methodologies

2.1 Data Partition and Migration on Distributed Memory Multiprocessors

Problem: Efficiency of parallel implementation depends on the implementation of the data movements that describe the required communication, since the overhead in distributed computing is in the required communication between processors. Although there are algorithms which address the complexity in data flow, in addition to arithmetic complexity, there is lack in unified methodology for analyzing and designing the data movements.

Approach: To present a formal methodology for the process of data distribution and redistribution using tensor products and stride permutations as tools. The algebraic expressions representing data partition and migration directly operate on data vectors, hence can be immediately embedded into an algorithm.

Goals: To implement and embed data partition and migration algorithms.

Applications: General numerical solutions that require successive stages of computation and data redistributions.

results: A unique data distribution technique that effectively uses transpose algorithms for multiplication of two rectangular matrices is derived and implemented. Performance of these algorithms are evaluated by carrying out implementations on Intel's *i860* based *iPSC/860*, Touchstone Delta, Gamma, and Paragon supercomputers.

Implemented the data redistribution algorithm for Euler partial differential equation (PDE) for two-dimensional case using wavelet-Galerkin method, where the two most important computation modules in this solution require two different data-partitions for their optimal implementation. Results of implementation on overall performance is included.

2.2 Efficient Multidimensional DFT Module Implementation on the INTEL i860 Processor

Problems: A standard method [2] of implementing non-power-of-2 transform size DFT is zero-padding. In multi-dimensional DFT computation, this will increase the transform size dramatically, not only slowing down the computation but also causing cache thrash and memory overflow. In the case of the parallel computer iPSC/860, each node processor has 8M byte memory. If the size of complex data to be processed is $72 \times 72 \times 72 = 373,248$, computation is made in the local memory of the processing unit without data segmentation. On the other hand, by padding with zeros, the size of complex data to be processed will be $128 \times 128 \times 128 = 2,097,152$, which is beyond the capacity of local memory; segmentation and data loading in and out will cause severe problem.

Approach: By formulating various DFT algorithms in the language of tensor products, any large size Fourier transform is built up by a collection of small size DFT modules which include as parameters decimation step sizes and twiddle factors. These parameters are introduced in the DFT modules to take advantage of modern computer architectures with parallel, pipelined, multi-functional structures, while providing flexibility into the building blocks.

Our library of core computation modules has the following features:

- We have efficiently implemented prime factors 3, 5, 7, 11, 13, 17 as well as powers of 2. Thus, transform size on each dimension of a multi-dimensional Fourier transform can have factors other than 2.
- One-dimensional small modules take advantage of vector operations on i860 by looping on other factors of the same dimension and other dimensions.
- One-dimensional small modules have pre-calculated twiddle factor array as a parameter. This provides for intermediate stages of Cooley-Tukey FFT implementation.

Goals: To create a scalable DFT library on the Intel i860 with mixed radix transform sizes with CPU time comparable to that of closest to a power of 2 transform size.

Results: Timing results of some sample medium size of 2-dimensional DFT modules with prime factor on each dimension is provided on the Intel i860 processor. The results of comparable power of 2 FFT package [6] that are commercially available are also included.

2.3 Efficient parallel implementation of traditional FFT codes

Problem: Data partition and migration for efficient communication in distributed memory architectures are critical for permance of data parallel programs.

Approach: Data partition and migration for efficient communication in distributed memory architectures are critical for performance of data parallel programs. This research presents a formal methodology for the process of data distribution and redistribution using tensor products and stride permutations as mathematical tools. The algebraic expressions representing data partition and migration directly operate on a data vector, and hence can be conveniently embedded into an algorithm. It is also shown that these expressions are useful for a clear understanding and to efficiently interleave problems that involve different data distributions at different phases. This compatibility made us successfully utilize these expressions in developing and demonstrating matrix transpose and fast Fourier transform algorithms. An endeavor to minimize communication cost using expressions for data distribution disclosed routing scheme for Fourier transform evaluation. Results promised that for large parallel machines, this scheme is a solution to today's problems which feature enormous data. Finally, a unique data distribution technique that effectively uses transpose algorithms for multiplication of two rectangular matrices is derived. Performance of these algorithms are evaluated by carrying out implementations on Intel's *i860* based *iPSC/860*, Touchstone Delta, Gamma, and Paragon supercomputers.

The global transposition stage, that interchanges the last two dimensions of the distributed among the processors data matrix, is interleaved with 1D FFT computations along the dimension that is orthogonal to the other two, to hide the communication cost and achieve a much better processor utilization.

In the 2D row-column FFT case, the global transposition can be decomposed into a number of smaller global transpositions of partial data that can be performed concurrently with the first stage of FFT computations (1D FFTs along the rows). In a similar fashion, the second global transposition step that is required if the results are to be returned in their original order, can be interleaved with the second FFT computational stage to totally hide the communication costs within the computations. For a more detailed description of the approach we followed please see Appendix B.

Goals: Create an efficient global transposition algorithm that interleaves computations with communications. Take full advantage of iPSC/860 hardware that allows to perform computations at the same time with performing communications, such that the data exchange stage starts at the same time with one of the computational stages of the 3D and 2D FFT.

Produce totally scalable efficient codes for large size multi-dimensional FFTs and evaluate their performance.

Applications: The power-of-two FFT has become a standard in many applications. The 3D FFT of large data size is a major component in a huge variety of signal processing applications in seismology, oil exploration, crystallography, meteorology, motion detection etc. The large-size 2D FFT has many applications ranging from image processing to system identification and signal reconstruction.

2.4 Vector-radix on the Paragon

The Vector-Radix (VR) algorithm is a vector generalization of the Cooley-Tukey algorithm for the case of two- and in general multi-dimensional FFT computations. In a uni-processor environment it has been shown that the VR can result in more efficient implementations than the straight forward application of the Row-Column (RC) method that computes a multi-dimensional FFT by sequentially applying 1D FFT along each of the dimensions, due mainly to the lower frequency of required accessing of a particular data point stored in the local memory than that required by the RC. In shared-memory multiprocessor systems, where the cost of data accessing is non-uniform, depending on where the data is stored, it is not clear that VR type of algorithms will be more efficient than RC method. The main advantage however of the VR formulation in the case of parallel multi-dimensional FFTs is the increased flexibility in initial and final data distribution and data/computations flow that allow for the design of codes that match in an optimal way the target multiprocessor machine parameters.

In the 2D case, VR formulations usually require three instead of two global communication stages which makes them unattractive for implementation on machines with high inter-processor communication costs. On the other hand, because the local memory accesses are much more regular than in the case of the RC implementations, for machines with fast inter-processor communication links, the VR results in more efficient implementations.

2.5 Reduced transform algorithms

Problems: Most variants of the Cooley-Tukey FFT algorithm deal with FFT computations as multiple stage calculations with data permutation between stages. This requires extensive interprocessor communication for implementing large size transpositions.

Approach: We present a strategy for computing a multidimensional DFT that hybrids a relatively new algorithm (Reduced Transform Algorithm) with already implemented single processor kernel routines. We will use the reduced transform algorithm to address the reduction and optimization of interprocessor communications. Our work has been mainly motivated from the distributed memory parallel computing paradigm, which is arguably the most difficult to harness due to its exposed interprocessor communication to the programmer. Most parallel computers require sophisticated algorithms and programming techniques for their optimum utilization. In this discussion, we will make use of algebraic facts in presenting the algorithms. The parameters in algebraic formulas give us the important implementation parameters. Thus the flexibility to address the variables in implementations is equated with flexibility in manipulating algebraic formalism. Initial investment in familiarity with some amount of algebra may be necessary, but the payoff is immediate. Most of the relevant algebra, not in its most rigorous form but its usage, can be found in [8].

In its most general form, the *Reduced transform algorithm* (RTA) is a full utilization of the duality between periodic and decimated data in the Fourier transform. This duality was used partially in some algorithms and implementations for restricted cases [4, 2, 5, 10]. A description of a generalization in a unified setting is found in [1, 9] along with the work of M. Rofheart [7]. In this paper, we will consider the application of RTA to the case $\mathbf{Z}/P \times \mathbf{Z}/P$, for a prime number P. Tensor product formulation of DFT computation on $\mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$ is interleaved with the periodization step in RTA for $\mathbf{Z}/P \times \mathbf{Z}/P$ to produce $P+1$ independent data of size NP.

We use the RTA to address the imbalance between computation and communication rates in current distributed memory parallel machines by reducing communication between processors to collective patterns only (broadcast and combine) instead of the all-to-all communication patterns required in the global matrix transpose needed by the row-column (RC) implementations of multidimensional DFT's. Also, since fast algorithms for prime size 1D-DFT's exist [8] and the case $\mathbf{Z}/P \times \mathbf{Z}/P$ of the RTA is very efficient because its computation requires only $P+1$ 1D transforms (versus $2P$ for the row column method), our approach addresses the issue of storage reduction by providing additional transform size options. For example the ability to perform a 181×181 point 2D DFT means potential

storage savings up to 50% over the 256×256 case, along with the savings in computational time. The storage savings can be used for the optimization of the broadcasting step needed for the RTA, in environments with long communications latency.

Via the Chinese remainder theorem, we will extend our method to compute the 3-dimensional DFT on $\mathbf{Z}/N \times \mathbf{Z}/MP \times \mathbf{Z}/KP$, where N is an arbitrary integer, M and K are integers not divisible by P, for a prime P. We transform the data set to an equivalent 5D data set on $\mathbf{Z}/N \times \mathbf{Z}/M \times \mathbf{Z}/K \times \mathbf{Z}/P \times \mathbf{Z}/P$, and then employs the RTA on the last two indices to break the problem into smaller independent sub-problems that can be computed in parallel. Each sub-problem is associated with the computation of the value of the Fourier Transform along one *line* in the set $\mathbf{Z}/P \times \mathbf{Z}/P$ passing through the origin. These lines intersect only at the origin and cover the index space. When translated from the 5D data set back to the original 3D data, each line corresponds to a set of parallel lines covering the index space.

Three stages are needed to compute the values of the DFT along the lines: (1) Periodization stage, which consists of additions of data along lines perpendicular to a given line, (2) 3D Cooley-Tukey FFT and (3) P-point DFT. In a multiprocessor environment, each processor computes these three steps independently of the others thus allowing for maximum parallelism and efficiency. Moreover, the final data distribution among the processors is such as to permit further processing in a parallel fashion since every processor holds only results belonging to the same geometrical subset.

The proposed hybrid method (HRTA) can be used in applications such as the computation of motion from a sequence of images (multi-frame detection, MFD), a very important task in computer vision, HDTV and video telephony. Several methods for MFD have been proposed in the literature that are usually divided into two categories: *Time Domain* methods, that estimate the motion by processing the sequence of images directly, and the recently proposed *Frequency Domain* methods [3], [6] that processes the frequency contents of the images to estimate the velocity and trajectory of the moving components. The latter methods offer more robust detection and huge computational savings since the frequency domain representation of the 3D data (sequence of 2D images) is more compact than the equivalent time domain representation. With all the processors holding data belonging to different lines in the frequency domain, each processor can independently test for the presence of motion along its assigned direction.

2.6 Non-power-of-two scalable DFT library based on RTA variants

Problem: In many applications the data size is not a power of two such that zero padding has to be employed to use the efficient FFT algorithms. In the multi-dimensional case, zero padding increases tremendously the data size and hence the required computational time.

Approach: The recently proposed RTA is combined with the Good-Thomas factorization and the standard Cooley-Tukey FFT algorithm to give DFT algorithms that require a reduced amount of inter-processor communications at the expense of larger data storage needs and additional pre-processing stages. The Hybrid RTA variants as well as the implementation issues are described in detail in Appendix A.

Goals: To create a totally scalable non-power-of-two DFT library for 2D and 3D cases employing the concepts of the RTA. Investigate in detail the performance and the tradeoffs of the new algorithms and propose efficient hardware structures that would further improve the DFT codes.

Applications: The special structure of the RTA that computes the output of the DFT along particular geometrical subsets of the original index set can be used for the fast moving target tracking and recognition, as well as for digital video compression. The RTA variants are especially suitable for implementation on DSP multi-processor boards and clusters of workstations.

2.7 Symmetrized crystallographic parallel DFT algorithms

Problem: In many applications (crystallography, higher order spectra computations) the data have inherent redundancy due to symmetries in their structure. In most cases these symmetries can be expressed as group actions (affine or point groups). If efficient algorithms for the computation of the DFT of such data are desired, the inherent data symmetries need to be taken into account to result in both data reduction and computational savings. Although considerable work has been done in the computation of symmetrized DFTs, algorithms that can be implemented in a parallel machine need to be derived.

Approach: A group theoretic approach is taken to decompose the data set into *orbits* that are characterized by constant data value, and to perform a data reduction step by choosing only one representative data point for each such orbit. To take advantage of fast DFT routines, the representatives of the orbits are being covered with the minimum number of *lines* through the data space, and then RTA variants are being employed to compute the value of the DFT along these points efficiently. The algorithm is being generalized for a large collection of data sizes by employing the Chinese Remainder Theorem and the Good-Thomas permutation.

Goals: Theoretical study of symmetrized DFT algorithms suitable for implementation in parallel multi-processor machines. Development of a unified theory to treat all symmetries usually encountered in practical applications. Development of a general symmetrized DFT library for the Intel iPSC/860 and Paragon multiprocessors.

Applications: Determination of the structure of a crystal from X-ray diffraction data, efficient computation of higher order statistics for signal analysis and reconstruction for application to material science and protein crystallography.

2.8 Implementation of integer and rationally oversampled Weyl-Heisenberg coefficient computation

Problem: During the last four years powerful new methods have been introduced for analyzing Wigner transforms of discrete and periodic signals based on finite Weyl-Heisenberg (W-H) expansions. A recent work adapted these methods to gain control over the cross-term interference problem by constructing signal systems in time frequency space for expanding Wigner transforms from W-H systems based on Gaussian-like signals. The computational feasibility of the method depends strongly on the availability of efficient and stable algorithms for computing W-H expansion coefficients.

Approach: The finite Zak transform is established as a fundamental and powerful tool for studying critically sampled and rationally oversampled W-H systems and for designing algorithms for computing W-H coefficients for discrete and periodic signals. The role of the finite Zak transform is analogous to that played by the Fourier transform in replacing complex convolution computations by simple pointwise multiplication. In this new setting properties of W-H systems such as their spanning space and dimension can be determined by simple operations on functions in Zak space. This relationship will impact on questions of existence, parameterization and computation of W-H expansions.

Implementation results on single RISC processor of i860 and the PARAGON parallel multiprocessor system are given for sample sizes both of powers of 2 and mixed sizes with factors 2, 3, 4, 5, 6, 7, 8, 9. The algorithms described in this paper possess highly parallel structure and are especially suited in a distributed memory parallel processing environment. Timing results on single i860 processor and on 4- and 8-node computing systems show that real-time computation of W-H expansions is realizable.

Results: Implementation results on single RISC processor of i860 and the PARAGON parallel multiprocessor system are given for sample sizes both of powers of 2 and mixed sizes with factors 2, 3, 4, 5, 6, 7, 8, 9. The algorithms described in this paper possess highly parallel structure and are especially suited in a distributed memory parallel processing environment. Timing results on single i860 processor and on 4- and 8-node computing systems show that real-time computation of W-H expansions is realizable.

2.9 Porting parallel multi-dimensional FFT codes to the IBM SP2 shared-memory multiprocessor

Problem: Recent advances in hardware provide vast possibilities in machine variations. Expensive and time-consuming efforts in software development are often required for effective utilization of these advances. In particular, framework for designing algorithms that takes architectural variations become most urgent.

Approach: Tensor product formalism and the finite abelian group theory has been the major tool for our algorithm design and implementation. Although our codes have been optimized for the Touchstone systems, the flexibility of our design tool allowed us to re-use the software and algorithmic skeletons and simply recompile and relink it with the machine-specific interprocessor communication and one-dimensional FFT libraries. The availability of efficient one- and two-dimensional FFT codes from the ESSL/6000 Engineering and Scientific Subroutine Library, including both powers of 2 and non-power of 2 sizes allowed us to design general purpose parallel 2D and 3D FFT codes that can handle a wide range of sizes. We are currently in the process of porting more codes to the IBM SP2 including RTA and Vector-Radix based FFT algorithms.

Results: The parallel FFT codes have been successfully ported to the IBM SP2 multiprocessor system of the NAS NASA Research Center in less than a day. The NAS SP2 machine has 160 nodes, each having at least 128 Mbytes of main memory and 2 Gbytes of disk space. The SP2 nodes are based on the RS6000/590 workstation configuration that relies on the POWER2 multi-chip RISC processor equipped with two integer and two floating point computation units capable of achieving a peak performance in the order of 250 MFlops.

2.10 Parallel DFT codes for Clusters of Workstations

Problem: Clusters Of WorkstationS (COWS) are becoming very attractive as easily available alternatives to expensive parallel supercomputers for certain classes of problems. Due to the special nature of this form of parallel machines (workstations connected via a common ethernet cable), row-column methods that require a global transposition step that implies all-to-all communication are highly inefficient. On the other hand the RTA variants that require no inter-processor communication at the expense of preprocessing the data emerge as the only viable approach.

Approach: The variants of the RTA decompose the task of DFT computation to a number of subtasks that can be computed independently at the expense of preprocessing the data. The broadcasting of the data to all available processors can be implemented very efficiently on the ethernet bus topology, since all processors have access to the broadcasting medium. The details of the parallel RTA implementation on a cluster of workstations are described in detail in Appendix C.

Goals: Develop a set of parallel DFT codes for clusters of workstations that can be used when data sizes are large and computational speed is important. Investigate the efficiency and scalability of the codes and improve the loading/unloading of data/results. Investigating the tradeoffs between the granularity of the partitioning into subtasks and the amount of data pre-processing to choose the most efficient RTA variant for the particular implementation. Experiment with large clusters of workstations (100-200) and develop methods for the computation of Giga-size DFTs.

Applications: Developing efficient codes for clusters of workstations will allow the processing and analysis of data sets much larger than with the computers available today to advance the research and understanding in a variety of applications in biomedical engineering, image processing, systems identification etc.

References

[1] M. An, I. Gertner, M. Rofhear, and R. Tolimieri. Discrete Fast-Fourier Transforms: A Tutorial Survey. In *Advances in Electronics and Electron Physics*, volume 80, pages 2–69. Academic Press, 1991.

[2] I. Gertner. A New Efficient Algorithm to compute the Two-Dimensional Discrete Fourier Transform. *IEEE Transactions on ASSP*, 36:1036–1050, July 1988.

[3] A. Kojima, N. Sakurai, and J. Kishigami. Motion Detection using 3D-FFT Spectrum. In *ICASSP Int. Conf. Accoustics, Speech and Signal Proc.*, volume 5, pages 213–216, 1993.

[4] Auslander L., Feig E., and S. Winograd. New Algorithms for the Multidimensional Discrete Fourier Transform. *IEEE Transactions on ASSP*, 31:388–403, February 1983.

[5] H.J. Nussbaumer and P. Qualdalle. Fast Computation of Discrete Fourier Transforms using Polynomial Transforms. *IEEE Transactions on ASSP*, 27:169–181, 1979.

[6] Boaz Porat and Benjamin Friedlander. A Frequency Domain Algorithm for Multiframe Detection and Estimation of Dim Targets. *IEEE Trans. on PAMI*, 12:398–401, April 1990.

[7] M. Rofheart. *Algorithms and Methods for Multidimensional Digital Signal Processing.* PhD thesis, CUNY, 1991.

[8] R. Tolimieri, M. An, and C. Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, New York, 1989.

[9] R. Tolimieri, M. An, and C. Lu. *Mathematics of Multidimensional Fourier Transform Algoritms.* Springer-Verlag, New York, 1993.

[10] M. Vulis. The Weighted Redundancy Transform. *IEEE Transactions on ASSP*, 37:1687–1692, November 1989.

3 Implementation Results

3.1 Data partition and migration schemes

Data partition and migration for efficient communication in distributed memory architectures are critical for performance of data parallel programs. We have developed a formal methodology for the process of data distribution and redistribution in terms of tensor products and stride permutations. The algebraic expressions representing data partition and migration directly operate on a data vector, and hence can be conveniently embedded into an algorithm. It is also shown that these expressions are useful for a clear understanding and for efficiently embedding into problems that involve different data distributions at different phases. A unique data distribution technique that effectively uses transpose algorithms for multiplication of two rectangular matrices is derived. Performance of these algorithms are evaluated by carrying out implementations on Intel's i860 based iPSC/860, Touchstone Delta and Paragon supercomputers.

3.1.1 Matrix transpose algorithms in three data distributions

Results of transpose algorithms on Paragon

M	N	Row-Division ms	Col-Division ms	Mesh-Division ms
128	128	5.236	6.172	1.316
128	256	5.902	7.051	2.028
128	512	9.031	10.409	2.159
128	1024	12.356	15.312	3.866
256	128	5.501	6.665	1.825
256	256	8.283	9.746	2.301
256	512	11.483	14.027	4.018
256	1024	20.076	22.503	7.548
512	128	8.310	9.432	3.450
512	256	11.555	13.359	5.905
512	512	18.536	21.122	7.954
512	1024	39.628	38.529	16.434
1024	128	11.228	13.132	5.815
1024	256	17.526	20.616	10.631
1024	512	31.211	37.445	20.889
1024	1024	50.936	66.403	49.274

Results of transpose algorithms on Touchstone Delta

M	N	Row-Division ms	Col-Division ms	Mesh-Division ms
128	128	8.092	8.865	2.681
128	256	10.042	12.280	5.769
128	512	13.988	18.980	11.702
128	1024	23.909	33.014	20.018
256	128	10.065	12.016	5.041
256	256	14.228	18.150	11.554
256	512	23.030	31.237	20.088
256	1024	43.458	59.109	36.009
512	128	13.982	17.920	9.822
512	256	23.002	30.593	19.637
512	512	44.178	57.799	36.091
512	1024	95.145	114.215	79.681
1024	128	22.743	30.400	19.507
1024	256	42.197	57.171	36.109
1024	512	83.011	113.416	79.492
1024	1024	187.484	223.287	167.497

3.1.2 Switching data partition schemes in application

We consider the implementation of numerical solution to Euler partial differential equation (PDE) for two-dimensional case using wavelet-Galerkin method. The two most important computation modules in this solution require two different data-partitions for their optimal implementation. First module, *Helmholtz*, involves two-dimensional filtering with forward and inverse Fourier transform methods. The second module computes *Jacobian* that consists of numerous small intra-node matrix multiplications. The module Jacobian requires boundary data from other nodes, but upto the necessity for neighboring spatial regions to exchange data, choice of any data-partitioning shows ideal concurrency, with no sequential dependence of one processor's calculation on other's. Departure from ideal speedup in evolution of Jacobian arises because the elements on node boundaries must be shared by geometrically neighboring processors. Minimization of the elements on the boundaries minimizes the internode communication, leading to the most optimal parallel implementation.

Optimal implementation of Helmholtz requires the data distribution along rows or columns of the data array, while Jacobian requires the data distribution in 2-dimensional subarrays (mesh-division). Switching between row-division and mesh-division data-partitions is required to make use of the peak performances of these modules individually.

Two-dimensional double-precision complex FFT implementation results

(1) iPSC/860 library code, (2) Interface routines appended at input and output, (3) Algorithm-1, (4) Algorithm-2.

Problem Size	Nodes	Intel ms	Interface ms	Algorithm-1 ms	Algorithm-2 ms
32 × 32	4	0.06054	0.13752	0.12409	0.08476
	16	0.12427	0.25118	0.20137	0.13195
64 × 64	4	0.15091	0.31761	0.28070	0.23038
	16	0.13451	0.26424	0.23804	0.17571
	64	0.48014	0.72160	0.53387	0.39570
128 × 128	4	0.50754	0.96545	0.86153	0.76560
	16	0.24929	0.44145	0.42941	0.33604
	64	0.49421	0.76185	0.58775	0.43177
256 × 256	4	1.94816	3.43353	3.17574	2.91836
	16	0.60610	1.13566	1.15002	1.00119
	64	0.57530	0.94583	0.82859	0.64410
	256	1.96009	2.73886	1.66710	1.54402
512 × 512	4	8.58407	14.55625	13.08064	12.30499
	16	2.37530	4.07935	4.16807	3.81806
	64	1.09181	2.17609	1.92430	1.63670
	256	2.54740	2.90163	2.29605	1.96358

Timing results for 128×128 size vorticity computations

Nodes	Jacobian		Helmholtz			Total		
	row-D	Mesh	row-D	Mesh1	Mesh2	row-D	Mesh1	Mesh2
4	2.8317	2.7939	0.11216	0.18218	0.16298	2.9438	2.9761	2.9568
16	0.8128	0.7310	0.06094	0.09950	0.07688	0.8738	0.8305	0.8079
64	0.3095	0.1996	0.10510	0.12022	0.08916	0.4146	0.3198	0.2887

3.1.3 Parallel matrix multiplication algorithm for rectangular arrays

Many applications have numerical solutions in which required computation is presented as matrix operations. One of the most elementary operations involving matrices is multiplication of two matrices. However, since matrix multiplication requires substantially more data movement than most other operations, algorithms that address efficient data movement is crucial for effective implementation on concurrent computers.

We have reviewed and implemented an existing matrix multiplication algorithm that generates and accumulates partial results by moving multiplicands through a set of broadcasts and shifts. Two extreme cases of data decomposition strategies cases involve either only a set of broadcasts or only a set of shifts. We designed a different approach that replaces broadcasts or shifts by matrix transpose. Identification of shortcomings in the two extreme cases of broadcast-and-shift algorithm and the fact that dot product of two vectors result in a single element is the motivation for this new approach. Then, to overcome the hurdles in memory requirement, we modified the algorithm for efficient data manipulation with the aid of block transpose algorithm. We present evaluation of communication costs in broadcast-and-shift algorithm versus new approach and timing results of their implementations on Intel's Paragon, Touchstone Delta and iPSC/860.

3.1.4 Implementation results on matrix multiplication algorithm

Timing results for routing scheme in new method

N_1 N_2 N_3	2-nodes	4-nodes	8-nodes	16-nodes
32 512 32	0.495	1.049	2.294	4.870
64 512 64	0.801	1.827	3.348	4.970
128 512 128	2.238	4.375	5.775	8.953
256 512 256	7.107	12.377	16.724	22.357
512 512 512	27.340	44.108	57.234	67.113

Timing results for routing schemes in matrix multiplication algorithms on 16 node Paragon

N_1	N_2	N_3	B-S Algor.	New App.	Performance Improvement
128	128	32	11.811	5.384	119.35
128	128	64	9.769	7.589	28.73
128	128	128	10.313	9.290	11.02
256	128	32	12.108	7.538	60.63
512	128	32	15.429	9.330	65.37
1024	128	32	22.604	13.469	67.82
128	256	32	11.185	5.355	108.88
128	256	64	11.753	7.573	55.19
128	256	128	12.853	9.359	37.33
256	256	32	14.466	7.530	92.10
256	256	64	14.993	9.339	60.53
512	256	32	20.114	9.341	115.33
512	256	64	20.618	13.529	52.40
1024	256	32	37.661	13.518	178.59
128	512	32	15.005	5.273	184.58
128	512	64	16.127	7.511	114.71
128	512	128	18.651	9.336	99.78
128	512	256	22.296	13.496	65.21
256	512	32	20.647	7.571	172.70
256	512	64	21.557	9.360	130.31
256	512	128	24.351	13.468	80.81
512	512	32	32.073	9.333	243.65
512	512	64	32.874	13.487	143.74
1024	512	32	66.446	13.524	391.31
1024	512	64	54.994	23.743	131.63

Timing results for routing schemes in matrix multiplication algorithms on 16 node Touchstone Delta

N_1	N_2	N_3	B-S Algor.	New App.	Performance Improvement
128	128	32	11.742	7.265	61.62
128	128	64	12.848	11.661	10.79
256	128	32	18.404	11.661	57.82
512	128	32	31.486	21.037	49.67
128	256	32	19.511	7.325	166.67
128	256	64	21.919	11.669	87.84
128	256	128	26.588	21.128	25.84
256	256	32	32.423	11.641	178.52
256	256	64	34.938	21.032	66.12
256	256	128	39.837	39.226	1.56
512	256	32	59.338	20.973	182.93
512	256	64	61.808	39.322	57.18
128	512	32	34.936	7.302	378.44
128	512	64	39.797	11.674	240.90
128	512	128	49.139	21.112	132.75
128	512	256	68.786	39.276	75.13
128	512	512	109.143	75.203	45.13
256	512	32	61.808	11.672	429.54
256	512	64	66.710	21.108	216.04
256	512	128	76.199	39.185	94.45
256	512	256	96.222	75.287	27.81
512	512	32	114.326	20.887	447.35
512	512	64	119.239	39.229	203.96
512	512	128	128.646	75.244	70.97

Timing results for routing schemes in matrix multiplication algorithms on 16 node iPSC/860

N_1	N_2	N_3	B-S Algor.	New App.	Performance Improvement
128	128	32	26.504	18.586	42.60
256	128	32	44.936	30.932	45.27
512	128	32	80.056	55.328	44.69
128	256	32	47.235	19.506	142.15
128	256	64	53.787	30.849	74.35
128	256	128	65.542	55.086	18.98
256	256	32	82.382	30.829	167.22
256	256	64	89.350	53.987	65.50
512	256	32	152.873	55.152	177.18
512	256	64	159.157	102.409	55.41
128	512	32	88.714	18.271	385.54
128	512	64	101.255	31.236	224.16
128	512	128	124.579	55.067	126.23
128	512	256	185.685	101.661	83.65
128	512	512	304.817	198.436	53.61
256	512	32	159.439	30.950	415.15
256	512	64	171.299	55.532	208.47
256	512	128	197.331	101.436	94.54
256	512	256	245.848	198.612	23.78
512	512	32	300.573	53.400	462.87
512	512	64	312.586	102.097	206.17
512	512	128	339.101	199.487	69.97

3.2 Single i860 node codes: The building blocks

Delta nodes (40 MHz i860)

size	forward	inverse	size	forward	inverse
16	8	8	20	11	11
27	19	19	32	17	17
40	23	22	64	34	35
96	70	70	100	74	74
125	91	91	128	90	90
140	109	109	160	109	113
192	145	142	224	169	163
256	182	179	300	302	303
360	282	289	400	376	376
448	370	366	512	395	408
576	521	526	700	709	707
768	781	776	800	788	801
900	970	968	1024	1024	1021
1024	1024	1021	1125	1393	1394
1280	1762	1740	1440	1867	1883
1536	2047	2020	1600	2161	2140
1800	2623	2553	2048	3745	3707

Paragon nodes (50 MHz i860)

The corresponding timings for the Kuck & Associates power-of-two FFT routines are given for comparison purposes. For both codes, the timings have been performed using the same method.

size	Aware	Kuck & Associates	size	Aware	Kuck & Associates
12	5.87		15	8.56	
16	6.81	13.95	20	10.67	
21	13.69		24	11.30	
25	14.73		27	15.69	
28	13.12		32	14.02	30.61
35	19.73		36	16.73	
40	20.91		45	23.57	
48	29.72		49	31.37	
56	26.73		64	26.98	38.95
80	49.91		96	57.42	
112	67.57		128	70.88	68.95
144	80.15		160	121.52	
192	123.39		224	136.14	
256	149.47	126.26	288	202.74	
320	244.88		384	274.98	
448	283.60		512	312.87	271.11
576	365.86		640	535.47	
768	629.24		896	706.53	
1000	825.14		1024	753.09	645.56
1152	923.92		1280	1185.01	
1536	1279.45		1792	1536.50	
2048	1793.91	1484.94			

Complex, single precision 2D FFTs on Paragon

The routines are coded in Fortran calling upon small size DFT assembly modules. Cooley-Tuckey algorithm is used.

size	time (ms)	size	time (ms)
40 x 40	2.138	40 x 36	1.753
40 x 35	1.815	40 x 28	1.213
40 x 25	0.930	40 x 24	0.832
40 x 20	0.651	40 x 15	0.516
40 x 12	0.371	36 x 40	1.799
36 x 36	1.493	36 x 35	1.529
36 x 28	0.968	36 x 25	0.799
36 x 24	0.713	36 x 20	0.574
36 x 12	0.328	32 x 32	1.114
20 x 40	0.651	20 x 36	0.594
20 x 35	0.595	20 x 32	0.480
20 x 28	0.440	20 x 25	0.382
20 x 24	0.379	20 x 20	0.311
20 x 16	0.224	20 x 15	0.249
20 x 12	0.187	15 x 40	0.507
15 x 36	0.452	15 x 35	0.465
15 x 32	0.382	15 x 28	0.351
15 x 25	0.335	15 x 24	0.303
15 x 20	0.247	15 x 16	0.188
15 x 15	0.197	15 x 12	0.143
12 x 40	0.371	12 x 36	0.329
12 x 35	0.344	12 x 32	0.280
12 x 28	0.267	12 x 25	0.248
12 x 24	0.224	12 x 20	0.188
12 x 16	0.130	12 x 15	0.145
12 x 12	0.111		

Complex, single precision 2D RTA FFTs on Paragon

The routines are coded in Fortran calling upon small size DFT assembly modules. The Hybrid RTA (HRTA) algorithm is used. Results are given for sizes: $p2^m \times p2^n$ where p is a prime number (3,5 or 7). *time1* referes to the time computational time when the data are already permuted (CRT has been pre-applied) and the output is being obtained on the algebraic lines. *time2* refers to the time required when the input is in its natural order (column-wise) and so is the output.

size	time1 (ms)	time2 (ms)	size	time1 (ms)	time2 (ms)
24 x 24	2.54	3.47	48 x 48	8.41	22.27
96 x 48	16.07	52.18	96 x 96	31.99	100.10
192 x 96	63.72	205.68	192 x 192	127.75	429.32
384 x 192	262.48	895.22	384 x 384	600.77	1882.31
768 x 384	1206.76	3823.15	768 x 768	2672.44	7903.65
20 x 20	2.52	5.13	40 x 20	3.89	9.31
40 x 40	6.26	17.23	80 x 40	11.71	34.03
80 x 80	22.62	66.52	160 x 80	45.31	134.27
160 x 160	90.51	278.01	320 x 160	180.81	576.84
320 x 320	361.13	1181.26	640 x 320	773.75	2500.96
640 x 640	1741.80	5217.32	28 x 28	4.52	9.67
56 x 28	7.37	17.64	56 x 56	12.57	33.39
112 x 56	24.05	65.64	112 x 112	46.06	131.22
224 x 112	91.57	265.79	224 x 224	179.97	549.69
448 x 224	357.37	1140.63	448 x 448	729.56	2339.62
896 x 448	1584.03	4904.57			

3.3 Multi-processor codes

3.3.1 Complex, 1D single precision FFTs

The Cooley-Tuckey formulation is being used. The 1-D FFT computation is formulated as a 2-D FFT with intermediate twiddle factors multiplication. Three global transpositions are required for the in-place 1-D FFT and two if it is not required that the distribution of the results coincides with that of the data. Furtehrmore, only one global transposition is required if the initial distribution of the data is assumed to be in a strided (transposed) fashion. In the following table, all times are in $sec \times 10^{-3}$, $time1$ refers to the in-place version and $time2$ to the out-of-place version.

Timings on the Paragon

nodes	size	time1	time2	nodes	size	time1	time2
2	1024	4.08	2.99	2	2048	6.21	4.75
	4096	10.44	8.15		8192	19.36	15.98
	16384	38.23	31.50		32768	71.21	59.22
	65536	139.70	116.71		131072	284.12	237.17
	262144	573.62	486.25		524288	1264.26	1083.69
	1048576	2771.17	2401.54				
4	1024	6.42	4.45	4	2048	7.75	5.32
	4096	10.88	8.09		8192	15.87	12.01
	16384	25.89	19.86		32768	46.62	36.64
	65536	86.68	69.06		131072	165.19	134.03
	262144	327.76	268.21		524288	702.36	581.81
	1048576	1500.23	1268.73				
8	1024	13.46	9.00	8	2048	13.86	9.34
	4096	15.13	10.22		8192	17.19	12.00
	16384	24.42	17.97		32768	35.65	26.50
	65536	56.75	43.36		131072	102.13	79.65
	262144	190.52	150.55		524288	380.92	312.05
	1048576	799.12	664.72				
16	1024	27.05	18.18	16	2048	27.64	18.66
	4096	28.55	19.20		8192	29.82	20.10
	16384	32.45	22.57		32768	37.29	25.95
	65536	51.87	37.84		131072	74.69	55.59
	262144	118.99	91.76		524288	226.34	179.86
	1048576	436.60	357.36				
32	1024	57.18	38.11	32	2048	55.45	36.97
	4096	55.51	37.50		8192	56.91	38.53
	16384	59.45	39.94		32768	62.06	42.81
	65536	66.50	45.78		131072	76.30	53.68
	262144	106.95	77.71		524288	158.60	119.73
	1048576	276.05	216.75				

3.3.2 Complex, 2D single precision FFTs

The hypercube transpose algorithm is used for both implementations. In the Intel code, the data is being assumed to be distributed row-wise (C convention) and in the Aware codes the data are distributed column-wise (Fortran convention). Two sets of timings are being reported: For the first set of timings (*time1*), two global data transpositions are required so that the final distribution of the results is the same as the original data distribution. In the second (*time2*), the second global data transposition is being ommited.

The Intel code, originally designed for the iPSC/860 hypercube, is using synchronous communication calls (csend) whereas the Aware code uses asynchronous communication calls (isend). The Aware code breaks the global transposition stage into two partial global transpositions and that are being performed concurrently with one-dimensional FFTs on the nodes.

Timing results on the Caltech Delta

time1 = in place, ms, Intel time2 = transposed, ms, Intel

time1a = in place, ms, Aware time2a = transposed, ms, Aware

nodes	size	time1	time2	time1a	time2a
2	1024 x 512	2205	1706	1555	1041
	512 x 512	962	737	764	507
	512 x 256	433	332	360	231
	256 x 256	196	145	158	108
	256 x 128	99	72	85	56
	128 x 128	51	36	40	31
4	1024 x 1024	2227	1647	1760	1154
	1024 x 512	1014	750	878	572
	512 x 512	472	344	424	275
	512 x 256	230	165	194	126
	256 x 256	113	80	96	74
	256 x 128	61	44	55	36
8	2048 x 1024	2464	1869	1955	1282
	1024 x 1024	1040	749	966	680
	1024 x 512	514	364	481	328
	512 x 512	247	173	226	147
	512 x 256	131	90	118	75
	256 x 256	67	46	41	43

nodes	size	time1	time2	time1a	time2a
16	2048 x 2048	2728	2078	2292	1491
	2048 x 1024	1212	893	1184	719
	1024 x 1024	533	370	520	333
	1024 x 512	273	186	255	184
	512 x 512	133	90	134	82
	512 x 256	73	50	89	53
32	4096 x 2048	3406	2278	3116	1887
	2048 x 2048	1656	1099	1419	859
	2048 x 1024	702	490	694	414
	1024 x 1024	413	208	313	188
	1024 x 512	168	108	183	108
	512 x 512	89	60	129	69
64	4096 x 4096	3856	2669	3741	2213
	4096 x 2048	1753	1240	1770	1059
	2048 x 2048	1018	599	790	496
	2048 x 1024	440	272	475	242
	1024 x 1024	362	127	221	103
	1024 x 512	135	140	165	-

The hypercube transpose algorithm is used. The data is assumed to be distributed column-wise (Fortran convention). For the first set of timings (*time1*), two global data transpositions are required so that the final distribution of the results is the same as the original data distribution. In the second (*time2*), the second global data transposition is being ommited. The code is using synchronous communication calls and is based on the original example provided in the iPSC/860 manuals. For the non-power-of-two 1D FFTs, in-house codes (libdft.a) are being employed. The timings are being reported for only a limited number of cases. Other DFT sizes, as well as mixed DFT-FFT cases can be treated as well.

Timings on the Paragon (non-power-of-2 sizes)

time1 = in place, ms, Aware time2 = transposed, ms, Aware

nodes	size	time1	time2
2	224 x 224	83	64
	300 x 300	165	133
	360 x 360	209	164
	448 x 448	335	264
	576 x 576	562	445
	640 x 640	759	616
	800 x 800	1134	908
	900 x 900	1530	1234
	1280 x 1280	3239	2657
	1440 x 1440	3979	3238
4	224 x 224	51	38
	300 x 300	98	75
	360 x 360	121	91
	448 x 448	188	143
	576 x 576	309	238
	640 x 640	410	327
	800 x 800	610	478
	900 x 900	831	654
	1280 x 1280	1730	1379
	1440 x 1440	2107	1685
	1600 x 1600	2670	2145

nodes	size	time1	time2
4	1792 x 1792	3291	2642
8	224 x 224	36	24
	360 x 360	77	55
	448 x 448	110	81
	576 x 576	174	132
	640 x 640	226	175
	800 x 800	333	256
	1280 x 1280	901	715
	1440 x 1440	1104	870
	1600 x 1600	1392	1105
	1792 x 1792	1711	1380
16	224 x 224	32	21
	448 x 448	74	51
	576 x 576	111	78
	640 x 640	140	103
	800 x 800	199	147
	1280 x 1280	486	379
	1440 x 1440	599	463
	1600 x 1600	743	581
	1792 x 1792	908	711

3.3.3 Real-to-Hermitian, 2D single precision FFTs

The hypercube transpose algorithm is used. The data is assumed to be distributed column-wise (Fortran convention). The timings do not include the final transposition stage, so that the results are obtained distributed along the first dimension.

The code is using asynchronous communication calls and interleaved computation/communication is being used. Each node, partitions the local data into two subsets and performs 1D FFTs on one subset while transposing the other one.

Timings on the Paragon (power of 2 sizes): version i2

time = transposed, ms, Aware

nodes	size	time
2	128 x 128	11
	128 x 256	21
	256 x 256	41
	256 x 512	87
	512 x 512	201
	512 x 1024	455
	1024 x 1024	997
4	128 x 128	9
	128 x 256	15
	256 x 256	25
	256 x 512	48
	512 x 512	96
	512 x 1024	236
	1024 x 1024	515
	1024 x 2048	1061
8	128 x 128	11
	128 x 256	14
	256 x 256	21
	256 x 512	32
	512 x 512	54
	512 x 1024	117
	1024 x 1024	269
	1024 x 2048	554
	2048 x 2048	1219

nodes	size	time
16	128 x 128	20
	128 x 256	21
	256 x 256	24
	256 x 512	30
	512 x 512	44
	512 x 1024	74
	1024 x 1024	139
	1024 x 2048	293
	2048 x 2048	627
32	128 x 128	38
	128 x 256	40
	256 x 256	41
	256 x 512	45
	512 x 512	50
	512 x 1024	65
	1024 x 1024	102
	1024 x 2048	183
	2048 x 2048	325
64	256 x 256	80
	256 x 512	83
	512 x 512	89
	512 x 1024	95
	1024 x 1024	110
	1024 x 2048	140
	2048 x 2048	217

The hypercube transpose algorithm is used. The data is assumed to be distributed column-wise (Fortran convention). The timings do not include the final transposition stage, so that the results are obtained distributed along the first dimension.

The code is using synchronous communication calls and the hypercube transpose algorithm.

Timings on the Paragon (power of 2 sizes): version i1

time = transposed, ms, Aware

nodes	size	time	nodes	size	time
2	128 x 128	10	8	1024 x 2048	443
	128 x 256	20		2048 x 2048	959
	256 x 256	39		2048 x 4096	1728
	256 x 512	80		4096 x 4096	3929
	512 x 512	163	16	128 x 128	11
	512 x 1024	378		128 x 256	13
	1024 x 1024	818		256 x 256	15
	1024 x 2048	1696		256 x 512	23
	2048 x 2048	3576		512 x 512	34
4	128 x 128	7		512 x 1024	64
	128 x 256	12		1024 x 1024	126
	256 x 256	23		1024 x 2048	239
	256 x 512	46		2048 x 2048	479
	512 x 512	89		2048 x 4096	850
	512 x 1024	197		4096 x 4096	1997
	1024 x 1024	428	32	128 x 128	21
	1024 x 2048	887		128 x 256	21
	2048 x 2048	1855		256 x 256	23
	2048 x 4096	3351		256 x 512	26
8	128 x 128	7		512 x 512	32
	128 x 256	10		512 x 1024	50
	256 x 256	16		1024 x 1024	80
	256 x 512	28		1024 x 2048	142
	512 x 512	51		2048 x 2048	271
	512 x 1024	108		2048 x 4096	463
	1024 x 1024	225		4096 x 4096	997

3.3.4 Complex-to-Complex, 3D FFT

The hypercube transpose algorithm is used. The data is assumed to be distributed along the last dimension. For the first set of timings (*time1*), two global data transpositions are required so that the final distribution of the results is the same as the original data distribution. In the second (*time2*), the second global data transposition is being ommited. The code is using synchronous communication calls and is based on the original example provided in the iPSC/860 manuals. Synchronous communication calls are being used.

Timing results on the Delta (power of 2 sizes - single precision)

time1 = in place, ms, Aware time2 = transposed, ms, Aware

nodes	size	time1	time2
2	32 x 32 x 32	122	99
	64 x 32 x 32	230	187
	64 x 64 x 32	442	358
	64 x 64 x 64	843	682
	128 x 64 x 64	1653	1335
4	32 x 32 x 32	70	54
	64 x 32 x 32	133	103
	64 x 64 x 32	254	196
	64 x 64 x 64	489	373
	128 x 64 x 64	959	732
	128 x 128 x 64	1893	1441
8	32 x 32 x 32	43	30
	64 x 32 x 32	76	56
	64 x 64 x 32	143	105
	64 x 64 x 64	345	198
	128 x 64 x 64	550	386
	128 x 128 x 64	1047	759
	128 x 128 x 128	2021	1590
16	32 x 32 x 32	28	16
	64 x 32 x 32	41	25
	64 x 64 x 32	61	41
	64 x 64 x 64	101	72
	128 x 64 x 64	184	136
	128 x 128 x 64	347	263
	128 x 128 x 128	669	517
	256 x 128 x 128	1315	1037

The hypercube transpose algorithm is used. The data is assumed to be distributed along the last dimension. For the first set of timing results (*time1*), two global data transpositions are required so that the final distribution of the results is the same as the original data distribution. In the second (*time2*), the second global data transposition is being ommited. The code is using synchronous communication calls and is based on the original example provided in the iPSC/860 manuals. Synchronous communication calls are being used. The local data permutations are being performed by using Kuck & Associates library matrix transposition calls.

Timings on the Paragon (power of 2 sizes-single precision)

time1 = in place, ms, Aware time2 = transposed, ms, Aware

nodes	size	time1	time2
2	32 x 32 x 32	71	58
	64 x 32 x 32	139	113
	64 x 64 x 32	279	230
	64 x 64 x 64	561	461
	128 x 64 x 64	1114	919
4	32 x 32 x 32	47	36
	64 x 32 x 32	83	65
	64 x 64 x 32	154	124
	64 x 64 x 64	300	242
	128 x 64 x 64	583	474
	128 x 128 x 64	1165	953
8	32 x 32 x 32	29	20
	64 x 32 x 32	49	35
	64 x 64 x 32	91	68
	64 x 64 x 64	171	131
	128 x 64 x 64	325	259
	128 x 128 x 64	644	509
	128 x 128 x 128	1282	1023

nodes	size	time1	time2
16	32 x 32 x 32	27	16
	64 x 32 x 32	40	25
	64 x 64 x 32	61	41
	64 x 64 x 64	103	73
	128 x 64 x 64	188	139
	128 x 128 x 64	353	269
	128 x 128 x 128	684	531
	256 x 128 x 128	1343	1063
32	32 x 32 x 32	43	23
	64 x 32 x 32	47	26
	64 x 64 x 32	56	33
	64 x 64 x 64	82	52
	128 x 64 x 64	124	84
	128 x 128 x 64	215	155
	128 x 128 x 128	398	294
	256 x 128 x 128	740	567
	256 x 256 x 256	1427	1114

The hypercube transpose algorithm is used. The data is assumed to be distributed along the last dimension. For the first set of timing results (*time1*), two global data transpositions are required so that the final distribution of the results is the same as the original data distribution. In the second (*time2*), the second global data transposition is being ommited. The code is using synchronous communication calls and is based on the original example provided in the iPSC/860 manuals. Synchronous communication calls are being used. The local data permutations are being performed by using Kuck & Associates library matrix transposition calls. Since the library contains only double precision real transpositions, the double precision complex transpositions have been reformulated in terms of the available library functions.

Timings on the Paragon (power of 2 sizes-double precision)

time1 = in place, ms, Aware time2 = transposed, ms, Aware

nodes	size	time1	time2
2	32 x 32 x 32	141	118
	64 x 32 x 32	274	229
	64 x 64 x 32	539	451
	64 x 64 x 64	1069	894
4	32 x 32 x 32	82	66
	64 x 32 x 32	151	128
	64 x 64 x 32	291	237
	64 x 64 x 64	567	466
	128 x 64 x 64	1126	926
8	32 x 32 x 32	52	39
	64 x 32 x 32	90	70
	64 x 64 x 32	164	133
	64 x 64 x 64	307	246
	128 x 64 x 64	593	479
	128 x 128 x 64	1181	962

nodes	size	time1	time2
16	32 x 32 x 32	40	30
	64 x 32 x 32	59	45
	64 x 64 x 32	98	76
	64 x 64 x 64	177	138
	128 x 64 x 64	326	258
	128 x 128 x 64	624	501
	128 x 128 x 128	1228	998
32	32 x 32 x 32	48	27
	64 x 32 x 32	58	34
	64 x 64 x 32	82	53
	64 x 64 x 64	120	83
	128 x 64 x 64	204	148
	128 x 128 x 64	373	278
	128 x 128 x 128	687	532
	256 x 128 x 128	1511	1240

3.4 Vector Radix (VR) on the Paragon

3.4.1 2D Vector Radix (VR) on the Paragon

Implementation Results

We have implemented the parallel algorithms described in the previous section on an Intel Paragon multiprocessor system, that is based on the i860XR microprocessor and employs a mesh interconnection network. Optimized assembly-coded routines for the nodes include 1D FFTs, routines from BLAS and matrix transposition routines. The RC method can be made very efficient since optimized 1D FFT routines can be used. For the partial VR algorithm, the computation of the p-point FFTs (p is the number of nodes) is being performed either via optimized hand coded assembly routines that perform strided small-sized FFTs with twiddle factor multiplication, or by performing the butterflies explicitly using vectorized complex multiply-accumulate routines from the BLAS library.

In Table 1, we compare the RC and PVR implementations for a variety of test and machine sizes. Although the PVR method has not been fully optimized it performs generally better than the RC with the advantage being more evident for relatively small sized machine partitions. For more than 16 nodes, the PVR algorithm performs only slightly better than the RC, however substantial optimization can be performed.

In Table 3.4.1, we compare the Collect-Distribute (CD) implementation with the Full VR. In both implementations the 2D data are being distributed along both dimensions and the results are obtained in-place. Again, as in the case of the RC, the CD method has the advantage of using highly optimized 1D FFT routines, at the expense of increased data movements. Clearly, as we can see from Table 3.4.1, the FVR implementation is more efficient that the CD method, and additional optimization in the computation of the radix $p \times q$ FFTs is possible.

m	n	nodes	PVR	RC
256	256	2	83	90
256	512		162	190
512	512		390	400
512	1024		690	918
1024	1024		1581	2065
256	512	4	96	109
512	512		187	229
512	1024		371	495
1024	1024		829	1093
1024	2048		1729	2282
2048	2048		3584	4742
512	512	8	113	123
512	1024		210	267
1024	1024		449	582
1024	2048		900	1186
2048	2048		1853	2443
512	512	16	84	66
512	1024		140	127
1024	1024		254	260
1024	2048		484	522
2048	2048		973	1110
2048	4096		1945	2061
512	512	32	93	71
512	1024		119	104
1024	1024		189	185
1024	2048		318	334
2048	2048		542	608
2048	4096		1021	1115
4096	4096		2036	2087

Comparison of the partial Vector-Radix approach and the Row Column optimized implementation (execution times are in milliseconds).

m	n	nodes	FVR	CD
256	512	4	119	155
512	512		230	301
512	1024		464	606
1024	1024		951	1237
1024	2048		2111	2676
512	512	8	132	-
512	1024		249	-
1024	1024		487	-
1024	2048		1062	-
2048	2048		2180	-
512	512	16	81	99
512	1024		151	188
1024	1024		275	377
1024	2048		546	750
2048	2048		1104	1559
2048	4096		2402	3106

Table 1: Timings for the Full VR implementation (execution times are in milliseconds).

3.4.2 The 3D Vector-Radix Implementation on the Paragon

Several variations of the 3D VR algorithms have been implemented for a variety of machine sizes. The VR algorithm offers a larger flexibility in data and computation flows as well as initial and final data distribution. In the timing results reported next, data are assumed to be distributed along the last dimension. Since in the 3D case, the length of the 1D FFTs that have to be computed is in general considerably smaller than in the 2D case (assuming that data should have sizes such that they can fit into the processors local memory), efficient *vectorized* FFT routines have been written. Although these routines are coded in Fortran, when they are used to compute vectorized FFTs of highly rectangular data structures they perform substantially better than the optimized assembly coded library 1D FFT routines. The greater flexibility that the 3D VR algorithm offers as well as other improvements in inter-processor communication strategies resulted in codes that are more than twice as fast than the corresponding RC 3D codes especially for relatively small sized machine configurations.

Timing results

nodes	size	time1	time2
2	32 x 32 x 32	60 (141)	54 (118)
	64 x 32 x 32	125 (274)	112 (229)
	64 x 64 x 32	246 (539)	212 (451)
	64 x 64 x 64	516 (1069)	458 (894)
4	32 x 32 x 32	30 (82)	27 (66)
	64 x 32 x 32	61 (151)	50 (128)
	64 x 64 x 32	117 (291)	102 (237)
	64 x 64 x 64	239 (567)	200 (466)
	128 x 64 x 64	475 (1126)	414 (926)
8	32 x 32 x 32	16 (52)	12 (39)
	64 x 32 x 32	27 (90)	23 (70)
	64 x 64 x 32	48 (164)	43 (133)
	64 x 64 x 64	116 (307)	97 (246)
	128 x 64 x 64	229 (593)	191 (479)
	128 x 128 x 64	505 (1181)	382 (962)
16	64 x 64 x 64	87 (177)	70 (138)
	128 x 64 x 64	144 (326)	134 (258)
	128 x 128 x 64	265 (624)	242 (501)
	128 x 128 x 128	502 (1228)	462 (998)
32	128 x 64 x 64	111 (204)	89 (148)
	128 x 128 x 64	191 (373)	158 (278)
	128 x 128 x 128	318 (687)	286 (532)
	256 x 128 x 128	586 (1511)	538 (1240)

All the timing results reported are in miliseconds. For convenience, the timings for the Row-Column method implementation for the corresponding data sizes are givem in parentheses.

time1 = in place, ms, Aware time2 = transposed, ms, Aware

3.5 Implementation results on IBM SP2

*time*1 refers to the time required to perform a forward 2D FFT in which the node distribution of the output coincides with that of the input. *time*2 is the corresponding time when the results are obtained in a transposed fashion, i.e. they are obtained in nodes different than that where the data where originally stored. All times are being measured by using the *mclock*() function call and they are reported in miliseconds.

Size ($n \times m$)	nodes	time1 (ms)	time2 (ms)
1024×1024	4	380	280
1024×1024	8	190	150
1024×1024	16	110	80
1024×1024	32	60	40
1024×1024	64	40	30
2048×1024	8	410	310
2048×1024	16	210	160
2048×1024	32	130	90
2048×1024	64	80	50
2048×2048	16	450	310
2048×2048	32	240	170
2048×2048	64	150	100

Table SP2-1: Time required for forward 2D complex single precision FFT.

From the timings reported in Table SP2-1 we see that each global matrix transposition requires approximatelly 25 In the case of the in-place parallel 2D FFT (i.e. when the node distribution of the results coincides with that of the data), about 50 of the total time is needed for the inter-processor communication and local data transpositions. This suggests that substantial improvements could be achieved by usings asynchronous communication calls to interleave node computations with data communications.

In Table SP2-2 we report timings for the case of parallel 3D FFTs. Again, *time*1 refers to the "in-order" case and *time*2 refers to the "out-of-order" case.

Size ($n \times m \times k$)	nodes	time1 (ms)	time2 (ms)
$128 \times 128 \times 64$	4	390	270
$128 \times 128 \times 64$	8	210	140
$128 \times 128 \times 64$	16	130	70
$128 \times 128 \times 64$	32	80	40
$128 \times 128 \times 64$	64	60	20
$128 \times 128 \times 128$	8	420	270
$128 \times 128 \times 128$	16	230	150
$128 \times 128 \times 128$	32	150	90
$128 \times 128 \times 128$	64	100	80

Table SP2-2: Time required for forward 3D complex single precision FFT. Again as it for the case of the 2D parallel FFT, inter-processor communication requires a substantial part of the total FFT time.

3.6 RTA multi-processor codes

For this set of codes, each node is assumed to store in its local memory the whole data set. Each node, performs CRT and the corresponding periodization and then computes 3D DFT. The timings do not include the final data re-indexing stage.

nodes	size	time (ms)
4	192 x 192	34.79
	384 x 192	71.35
	384 x 384	162.25
	768 x 384	326.89
	768 x 768	715.19

3.7 Implementation results for Gabor coefficients

Table 1. Timing Results on i860 Single Node (Critical Sampling – 2^k)

Sample Size n	2-D $L \times M$	Time $ms = 10^{-3} sec.$
256	16×16	0.67
512	16×32	1.20
1024	32×32	2.02
2048	32×64	3.98
4096	64×64	7.41
8192	64×128	14.96
16384	128×128	29.82
32768	128×256	60.89
65536	256×256	125.55
131072	256×512	264.60
262144	512×512	566.99

Table 2. Timing on i860 Single Node (Critical Sampling – Mixed sizes)

Sample Size n	2-D $L \times M$	Time $ms = 10^{-3} sec.$
384	8×48	1.47
768	16×48	1.99
1536	32×48	3.12
3072	64×48	5.91
3072	128×24	6.15
6144	128×48	12.07
6144	64×96	12.48
12288	512×24	26.07
12288	128×96	24.05
24576	256×96	48.70
49152	256×192	98.71
98304	256×384	203.52
98304	512×192	209.12
196608	512×384	433.41
393216	1024×384	1011.61

Table 3. Timing Results on i860 Single Node (Integer Oversampled)

Sample Size $n = L'M$	2-D $L' \times M$	Time $ms = 10^{-3} sec.$
512	32×16	0.67
1024	32×32	1.20
2048	64×32	2.02
4096	64×64	3.98
8192	128×64	7.41
16384	128×128	14.96
32768	256×128	29.82
65536	256×256	60.89
131072	512×256	125.55
262144	512×512	264.60
524288	1024×512	566.99

Table 4. Timing on i860 Single Node (Fractional Oversampling (3/2))

Sample Size n	2-D $L \times M$	Time $ms = 10^{-3} sec.$
384	16×24	2.06
768	32×24	2.97
768	16×48	3.91
1536	64×24	5.31
1536	32×48	6.03
3072	64×48	10.79
3072	128×24	10.05
6144	128×48	20.85
6144	64×96	22.86
12288	128×96	43.15
24576	256×96	84.71
49152	256×192	171.39
98304	256×384	412.12
98304	512×192	413.50
196608	512×384	840.02

Table 5. Timing on i860 Single Node (Fractional Oversampling (5/4))

Sample Size n	2-D $L \times M$	Time $ms = 10^{-3} sec.$
320	8×40	2.82
640	16×40	3.85
1280	32×40	5.66
1280	16×80	7.66
2560	64×40	9.65
2560	32×80	11.35
5120	128×40	16.42
5120	64×80	18.32
5120	32×160	22.49
10240	128×80	32.09
10240	64×160	37.99
20480	128×160	67.65
20480	64×320	74.42
40960	128×320	134.08
81920	256×320	258.40
81920	128×640	276.69
163840	512×320	522.19
163840	256×640	534.90
327680	512×640	1149.76

A Formulating Data-Partition and Migration in Distributed Memory Multiprocessors

Abstract

This paper presents an algebraic framework for expressing data-partition and migration in distributed memory multiprocessors in terms of the algebra of stride permutations. This algebra provides powerful tools for visualizing the cost of communication in parallel computations and for minimizing this cost by straightforward algebraic manipulations. We demonstrate the significance of this tool and show how it leads to significant performance gains on Intel's Touchstone systems (Delta, iPSC/860 and Paragon) in three examples: matrix transpose algorithm, two-dimensional discrete Fourier transform algorithm, and solution of Euler partial differential equations using wavelet-Galerkin method.

A.1 Introduction

It is well known that data-distribution in distributed memory multiprocessors is essential to achieve high performance of data-parallel programs. Extensive research has been reported on data-decomposition optimization for distributed memory machines [1, 2, 3, 4, 5]. Research in this area can be crudely classified into two categories. One aims at finding optimal data-partitioning schemes for parallel loop constructs as part of compiler. It has been shown that the problem of finding an optimal data-partition is NP-complete [3, 6, 1]. Therefore, researchers have to rely on heuristic methods [6, 7, 8, 2, 9]. The other effort aims at special-purpose implementations and a large work force for developing optimal implementation of individual algorithms is reported [10, 11, 12].

Typically, an application requires a number of computation modules linked together to accomplish a specific computation. Global optimization depends not only on optimal implementation of the computational modules, but at least equally on the interface between these implementations as determined by the data partition and migration across processors.

In this paper, we present a systematic formulation for data-partition and migration on distributed memory multiprocessors in terms of tensor product notation and stride permutations. Data-partition and migration are represented using simple tensor algebraic expressions

highlighting the computational and communication complexity of parallel algorithms. Therefore, optimal data-partition and migration at interfaces between different algorithms becomes straightforward tensor algebraic manipulations with the aid of well-established theorems in this field. Furthermore, due to the conciseness of the underlying algebra, definitions are simple and compact without having to deal with complicated indices in complex data structures.

In order to demonstrate the significance and usefulness of our framework, we have carried out experiments on existing distributed memory multiprocessors such as Intel's Paragon, and Touchstone Delta. Initially, our formal definitions are incorporated in three application problems: matrix transpose algorithm, two dimensional discrete Fourier transform algorithm, and solution of Euler partial differential equation using wavelet-Galerkin approach. Then, simple algebraic manipulations on these expressions are carried out to derive optimal data-partition and migration schemes. Experimental timing results on these machines show that such simple algebraic manipulations result in performance improvement ranging from 30% to 600%.

The rest of the paper begins with a simple introduction to tensor notation and stride permutations as a background of our work. In Section 3, we present our formal definitions for data-partition and migration in distributed memory multiprocessor systems. Experiments on Intel's distributed machines and discussions on numerical results are presented in Section 4. Section 5 discusses the related work in the field with respect to our model. We conclude the paper in Section 6.

A.2 Preliminaries

In this subsection, we review and describe necessary notation and terminology that will be used throughout the paper.

A.2.1 Stride Permutations

A vector $\mathbf{x}$ is an ordered finite linear array. The dimension of $\mathbf{x}$, denoted by $dim(\mathbf{x})$, is the number of elements in the linear array. Let $dim(\mathbf{x}) = LS$, for positive integers L and S. Stride permutations are natural way of representing data-shuffling operations. We use $P(LS, S)$ to represent the stride permutation operation on a vector of length LS with stride S. To define $P(LS, S)$, set

$$\mathbf{y} = P(LS, S)\mathbf{x}.$$

The first L elements of $\mathbf{y}$ are obtained by collecting elements of $\mathbf{x}$ starting at element x_0 and then striding through $\mathbf{x}$ in steps of size S, i.e., $[x_0, x_S, \ldots x_{(L_s-1)S}]$. The next L elements of $\mathbf{y}$ are obtained in the same way starting at x_1 of $\mathbf{x}$: $[x_1, x_{S+1}, \ldots, x_{(L_s-1)S+1}]$, and so on. We can represent the stride permutation $P(LS, S)$ by a permutation matrix which we will denote also by $P(LS, S)$.

Example A.1 *Permutation matrix* $P(6,3)$ *operating on vector* $\mathbf{x} = [x_0\ x_1\ x_2\ x_3\ x_4\ x_5]^T$, *is given by*

$$P(6,3)\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{1}$$

A.2.2 Tensor Product

Tensor product is a binary operator between two matrices of any size. Given two matrices $\mathbf{A}$ and $\mathbf{B}$ of sizes $M_A \times N_A$ and $M_B \times N_B$, respectively, a new matrix, $\mathbf{C}$, dimensioned $M_A M_B \times N_A N_B$ can be generated by tensor product of $\mathbf{A}$ and $\mathbf{B}$ as:

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{(0,0)}\mathbf{B} & a_{(0,1)}\mathbf{B} & a_{(0,2)}\mathbf{B} & \cdots & a_{(0,N_A-1)}\mathbf{B} \\ a_{(1,0)}\mathbf{B} & a_{(1,1)}\mathbf{B} & a_{(1,2)}\mathbf{B} & \cdots & a_{(1,N_A-1)}\mathbf{B} \\ a_{(2,0)}\mathbf{B} & a_{(2,1)}\mathbf{B} & a_{(2,2)}\mathbf{B} & \cdots & a_{(2,N_A-1)}\mathbf{B} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{(M_A-1,0)}\mathbf{B} & a_{(M_A-1,1)}\mathbf{B} & a_{(M_A-1,2)}\mathbf{B} & \cdots & a_{(M_A-1,N_A-1)}\mathbf{B} \end{bmatrix}, \tag{2}$$

where $a_{(i,j)}$ is the element on the ith row and jth column of $\mathbf{A}$, and $a_{(i,j)}\mathbf{B}$ is scalar-matrix multiplication.

Example A.2 *Consider the following two matrices:*

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \textit{and} \quad \mathbf{B} = \begin{bmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \end{bmatrix}.$$

Then

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \left[\begin{array}{c|c} \mathbf{B} & 2\mathbf{B} \\ \hline 3\mathbf{B} & 4\mathbf{B} \end{array}\right] = \left[\begin{array}{ccc|ccc} 10 & 11 & 12 & 20 & 22 & 24 \\ 13 & 14 & 15 & 26 & 28 & 30 \\ \hline 30 & 33 & 36 & 40 & 44 & 48 \\ 39 & 42 & 45 & 52 & 56 & 60 \end{array}\right],$$

according to equation (2).

Two types of tensor products are of special interest to us here. One has an identity matrix on the left-hand side of a tensor product, called *prior identity matrix*, and the other has an identity matrix on the right-hand side, referred to as *post identity matrix.*
Denote the $N \times N$ identity matrix by I_N. For an $M \times M$ matrix A, $I_N \otimes A$ denotes the $MN \times MN$ block-diagonal matrix

$$\begin{bmatrix} A & & & \\ & \cdots & & \\ & 0 & \cdot\ 0 & \\ & & \cdot & \\ & & & A \end{bmatrix}.$$

Example A.3 *Consider a 4-processor machine and the butterfly matrix* $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Then, $\mathbf{y} = (\mathbf{I}_4 \otimes \mathbf{A})\ \mathbf{x} =$

$$\left[\begin{array}{c} x_0 + x_1 \\ x_0 - x_1 \\ \hline x_2 + x_3 \\ x_2 - x_3 \\ \hline x_4 + x_5 \\ x_4 - x_5 \\ \hline x_6 + x_7 \\ x_6 - x_7 \end{array}\right] = \left[\begin{array}{cccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{array}\right] \left[\begin{array}{c} x_0 \\ x_1 \\ \hline x_2 \\ x_3 \\ \hline x_4 \\ x_5 \\ \hline x_6 \\ x_7 \end{array}\right]$$

Each processor executes one butterfly on a different part of $\mathbf{x}$*, where the node boundaries are represented by horizontal lines. If only 2 processors are available, then we can use the identity*

$$I_4 \otimes A = I_2 \otimes (I_2 \otimes A)$$

to implement the computation where two butterflies are performed in each processor.

Tensor products with prior identity matrices can be used to represent parallel tasks. In general, on a k-processor distributed memory machine, execution of $(\mathbf{I}_N \otimes \mathbf{A})$ would imply k parallel tasks, where $N = nk$ and n is a positive integer.

If an identity matrix appears on the right-hand side of a tensor product (post identity matrix) it is performed in a natural way as a vector operation.

$$A \otimes I_N = \begin{bmatrix} a_{(0,0)}\mathbf{I}_N & a_{(0,1)}\mathbf{I}_N & a_{(0,2)}\mathbf{I}_N & \cdots & a_{(0,K-1)}\mathbf{I}_N \\ a_{(1,0)}\mathbf{I}_N & a_{(1,1)}\mathbf{I}_N & a_{(1,2)}\mathbf{I}_N & \cdots & a_{(1,K-1)}\mathbf{I}_N \\ a_{(2,0)}\mathbf{I}_N & a_{(2,1)}\mathbf{I}_N & a_{(2,2)}\mathbf{I}_N & \cdots & a_{(2,K-1)}\mathbf{I}_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{(L-1,0)}\mathbf{I}_N & a_{(L-1,1)}\mathbf{I}_N & a_{(L-1,2)}\mathbf{I}_N & \cdots & a_{(L-1,K-1)}\mathbf{I}_N \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{L_x-1} \end{bmatrix}. \quad (3)$$

Example A.4 *Consider a vector computer with vector register length equal to 3, and operational matrix defined as:*

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \textit{and} \quad \mathbf{x} = [x_0\ x_1\ x_2\ x_3\ x_4\ x_5]^T.$$

Then, $\mathbf{y} = (\mathbf{A} \otimes \mathbf{I}_3)\ \mathbf{x} =$

$$\begin{bmatrix} ax_0 + bx_3 \\ ax_1 + bx_4 \\ ax_2 + bx_5 \\ cx_0 + dx_3 \\ cx_1 + dx_4 \\ cx_2 + dx_5 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & b & 0 & 0 \\ 0 & a & 0 & 0 & b & 0 \\ 0 & 0 & a & 0 & 0 & b \\ c & 0 & 0 & d & 0 & 0 \\ 0 & c & 0 & 0 & d & 0 \\ 0 & 0 & c & 0 & 0 & d \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix},$$

is performed by partitioning input data into two subvectors $\mathbf{x}_1 = [x_0\ x_1\ x_2]^T$ *and* $\mathbf{x}_2 = [x_3\ x_4\ x_5]^T$, *and with the vector operations:* $\mathbf{y}_1 = a\mathbf{x}_1 + b\mathbf{x}_2$ *and* $\mathbf{y}_2 = c\mathbf{x}_1 + d\mathbf{x}_2$.

A.2.3 Some Useful Theorems

Tensor product identities provide powerful tool developing variants of an algorithm. We will present these properties without proofs, for there are many texts containing the proofs on diverse levels including [13, 14]. We use the convention that a complex tensor product formulation should be read from right to left.

Theorem A.1 Multiplication of Tensor Products: *If $N_X = M_A$ and $N_Y = M_B$, then the following multiplication theorem holds true.*

$$\begin{aligned}(\mathbf{X}_{M_X \times N_X} \otimes \mathbf{Y}_{M_Y \times N_Y}) \quad & (\mathbf{A}_{M_A \times N_A} \otimes \mathbf{B}_{M_B \times N_B}) \\ = \quad & (\mathbf{X}_{M_X \times N_X}\ \mathbf{A}_{M_A \times N_A}) \otimes (\mathbf{Y}_{M_Y \times N_Y}\ \mathbf{B}_{M_B \times N_B})\end{aligned} \tag{4}$$

This theorem is quite often used to derive parallel or vector computations when identity matrices appear in the product.

Theorem A.2 Commutative Law:

$$(\mathbf{A}_{M_A \times N_A} \otimes \mathbf{B}_{M_B \times N_B}) = P(M_A M_B, M_A)\,(\mathbf{B}_{M_B \times N_B} \otimes \mathbf{A}_{M_A \times N_A})\,P(N_A N_B, N_B) \tag{5}$$

This theorem is quite useful in generating different communication structures of an algorithm.

Theorem A.3 Inverse of Tensor Products: *Unlike the case in multiplication of two matrices, inverse of tensor product of two matrices does not change the order of its parameters.*

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \left(\mathbf{A}^{-1} \otimes \mathbf{B}^{-1}\right) \tag{6}$$

Theorem A.4 Multiplication Theorem of Stride Permutations: *Any simple-stride permutation can be decomposed into two stride permutations when stride is a multiple of two integers.*

$$P(N_1 N_2 N_3, N_1 N_2) = P(N_1 N_2 N_3, N_1) P(N_1 N_2 N_3, N_2) \tag{7}$$

Theorem A.5 Inverse Stride Permutation:

$$P(N_1 N_2, N_1)^{-1} = P(N_1 N_2, N_2). \tag{8}$$

Theorem A.6 Parallel-Vector Tensor Factorization of Stride Permutations:

$$P(N_1 N_2 N_3, N_3) = [P(N_1 N_3, N_3) \otimes \mathbf{I}_{N_2}]\,[\mathbf{I}_{N_1} \otimes P(N_2 N_3, N_3)] \tag{9}$$

This is one of the very important theorems to uncover the extent of communication complexity hidden in a permutation. When parameter N_1 is an integral multiple of number of processing elements, this theorem extracts parallel local operations from operations that depend upon non-local data. A stride permutation can also be factored in a different way (inverse of theorem (A.6)) leading to the following theorem.

Theorem A.7 Vector-Parallel Tensor Factorization of Stride Permutations:

$$P(N_1 N_2 N_3, N_1 N_2) = [\mathbf{I}_{N_1} \otimes P(N_2 N_3, N_2)]\,[P(N_1 N_3, N_1) \otimes \mathbf{I}_{N_2}] \tag{10}$$

A.3 Data Partition and Migration: Formal Definitions

A.3.1 Storing Data in Distributed Memories

Most large scale applications of scientific computing involve manipulations of data that are expressed in terms of matrices and vectors. This is natural because matrix notation gives a compact way to express computation. Moreover, storing matrices or vectors in the memory of a computer system is the first step of any computation. Different ways of storing data may result in different algorithmic structures as well as different computational performance. While methodology and algebraic formulations for storing matrices in a linear memory space of a single processor system exist, such as row-major and column-major, there is neither a formal and commonly agreed way of addressing data stored in distributed memory multiprocessor systems, nor an agreed formal description for various storage schemes. Programmers for parallel machines usually organize data in a way based on their convenience and efficiency of a specific algorithm. As a result, data-allocation and partition in parallel processing are very diversified. Therefore, there is a need for a unified approach for formalizing data allocation and partitioning in parallel machines, and for a clear and convenient mathematical representation of various data-storage schemes. In parallel computers, particularly in distributed memory multiprocessors, communication costs are directly related to various data storage schemes. Clear representation of storage schemes helps parallel programmer greatly to look into structures of implementations and communication costs associated with algorithms.

Consider a message-passing multiprocessor system with k processors labeled from 0 to $k-1$, where $k = k_1\ k_2$. We would like to partition and store a two-dimensional (2D) matrix, denoted by $\mathbf{A}$ onto this system. For the purpose of simplicity and clarity of our presentation, we present only the cases where the data can be evenly divided into k subsets and concentrate on our main interest of algebraic representation of partitioning the matrix and storing them into processors' memories. In the following, we assume that the operator $Vect_{MN}(\mathbf{A})$ maps an $M \times N$ size two-dimensional array, $\mathbf{A}$, into a MN-length single dimension array, $\mathbf{a}$, where (i,j)th element of $\mathbf{A}$ is mapped to $(j-1)\ M + i$th element of $\mathbf{a}$ (column-major).

Definition A.1 **Row-Division:** *Let* $\mathbf{A}$ *be an* $M \times N$ *matrix. We define row-division onto* k *processors as follows. Partition* $\mathbf{A}$ *into* k *sets of complete rows such that* i*-th set of rows (top-down) is allocated to* i*-th processor. In matrix notation, row-division can be represented as operating by*

$$\mathbf{P}_R(M,N,k) = P(Nk,k) \otimes \mathbf{I}_{M/k} \tag{11}$$

on a vector **a** *that is formed as* $Vect_{MN}(\mathbf{A})$.

We use bold faced "P" (**P**) with appropriate subscript to represent our data-partition definitions while italic "P" (P) to represent operation of stride permutation explained in section (A.2.1).

Definition A.2 Column-Division: *Let* **A** *be an* $M \times N$ *matrix. We define column-division onto k processors as follows. Partitioning matrix* **A** *into k sets of complete columns such that i-th set of columns (left-right) is allocated to i-th processor. In matrix notation, column-division is represented as operating by*

$$\mathbf{P}_C(M, N, k) = \mathbf{I}_{MN} \tag{12}$$

on a vector **a** *that is formed as* $Vect_{MN}(\mathbf{A})$.

Definition A.3 Mesh-Division: *Let* **A** *be an* $M \times N$ *matrix. We define mesh-division of* **A** *onto a system of* $k_1 \times k_2$ *processors as follows. Partition* M *rows of* **A** *into* k_1 *equal sets of rows (top-down) and then partition each set of rows into* k_2 *equal subsets (left-right). Each subset is a* $M/k_1 \times N/k_2$ *size matrix but will have neither complete rows nor complete columns. Allocation of these subsets to k processors is performed anti-lexicographically (top-down and then left-right). In matrix notation, mesh-division is defined as*

$$\mathbf{P}_M(M, N, k_1, k_2) = \mathbf{I}_{k_2} \otimes P(Nk_1/k_2, k_1) \otimes \mathbf{I}_{M/k_1}. \tag{13}$$

Definition A.4 Cyclic-Division: *Let* **A** *be an* $M \times N$ *matrix. We define cyclic-division of* **A** *onto k processors as follows. Partition the vector* $Vect_{MN}(\mathbf{A})$ *into* (MN/k) *consecutive subvectors such that i-th element of each subvector is allocated to i-th processors. In matrix notation, cyclic-division can be represented as operating by*

$$\mathbf{P}_{Cyc}(M, N, k) = P(MN, k) \tag{14}$$

on a vector **a** *that is formed as* $Vect_{MN}(\mathbf{A})$.

Definition A.5 Block-Cyclic-Division: *Let* **A** *be an* $M \times N$ *matrix. We define block-cyclic-division of* **A** *onto a system with k processors as follows. Partition the vector* $Vect_{MN}(\mathbf{A})$ *into* (MN/S) *number of S-length consecutive subvectors and assign* $[i \pmod k]$*-th subvector to i-th processor. In matrix notation, block-cyclic-division can be represented as operating by*

$$\mathbf{P}_{BC}(M, N, k) = P(MN/S, k) \otimes \mathbf{I}_S \tag{15}$$

on a vector **a** *that is formed as* $Vect_{MN}(\mathbf{A})$.

Block-cyclic is similar to cyclic except that each time S elements are allocated to a processor instead of one element. Also note that column-division can be obtained from block-cyclic-division for the case of $M/k = S$, that is, number of rows assigned to each processor in matrix $\mathbf{A}$ is equal to the length of subvector in block-cyclic-division.

Following five equations represent inverse operations of the above five definitions which can be derived using theorems (A.3) and (A.5).

$$\mathbf{P}_R^{-1}(M,N,k) = P(Nk,N) \otimes \mathbf{I}_{M/k} \tag{16}$$

$$\mathbf{P}_C^{-1}(M,N,k) = \mathbf{I}_{MN} \tag{17}$$

$$\mathbf{P}_M^{-1}(M,N,k_1,k_2) = \mathbf{I}_{k_2} \otimes P(Nk_1/k_2, N/k_2) \otimes \mathbf{I}_{M/k_1} \tag{18}$$

$$\mathbf{P}_{Cyc}^{-1}(M,N,k) = P(MN, MN/k) \tag{19}$$

$$\mathbf{P}_{BC}^{-1}(M,N,k) = P(MN/S, MN/(Sk)) \otimes \mathbf{I}_S \tag{20}$$

Example A.5 *This example demonstrates data-partitioning of an* 8×8 *matrix,* $\mathbf{A}$*,onto a 4-processor machine. Figure 1 shows how a 64-element vector* $\mathbf{a}$ *formed by* $Vect_{64}(\mathbf{A})$ *is partitioned in row-division, column-division, and mesh-division based on definitions (A.1)-(A.3). In case of row-division,* $\mathbf{I}_2$ *on the right-hand side of* $\mathbf{P}_R(8,8,4)$ *represents moving vectors of length 2 according to the permutation matrix* $P(32,4)$*. When this permutation is applied, resulting data at processor-0 is shown with dotted-line. For column-division data-partitioning, since input permutation is an identity matrix, no action needs to be performed, and the vector* $\mathbf{a}$ *is just segmented into four parts for allocating to four processors. For mesh-division data-partitioning,* $\mathbf{I}_2$ *on the left-hand side of* $\mathbf{P}_M(8,8,2,2)$ *represents an action to divide the vector* $\mathbf{a}$ *into two equal sets and perform the vector-stride action* $P(8,2) \otimes \mathbf{I}_4$ *on each set. However, this vector-stride further divides each set into eight small subvectors of length 4 and shuffle them according to the permutation* $P(8,2)$*. Once again, data residing at processor-0 after the action of input permutation is shown with dotted-line.*

General Usage of Data-Partition Definitions

Consider any computational procedure that is expressed by an operational matrix $\mathbf{G}$ operating on a vector $\mathbf{a}$ to obtain vector $\mathbf{b}$:

$$\mathbf{b} = \mathbf{G}\,\mathbf{a}. \tag{21}$$

This equation ignores the underlying data-partition necessary to carry out the computation in distributed memory multiprocessor system. To bring out the data-partition, let $\hat{\mathbf{a}}\,(= \mathbf{Q}_1\mathbf{a})$ be a

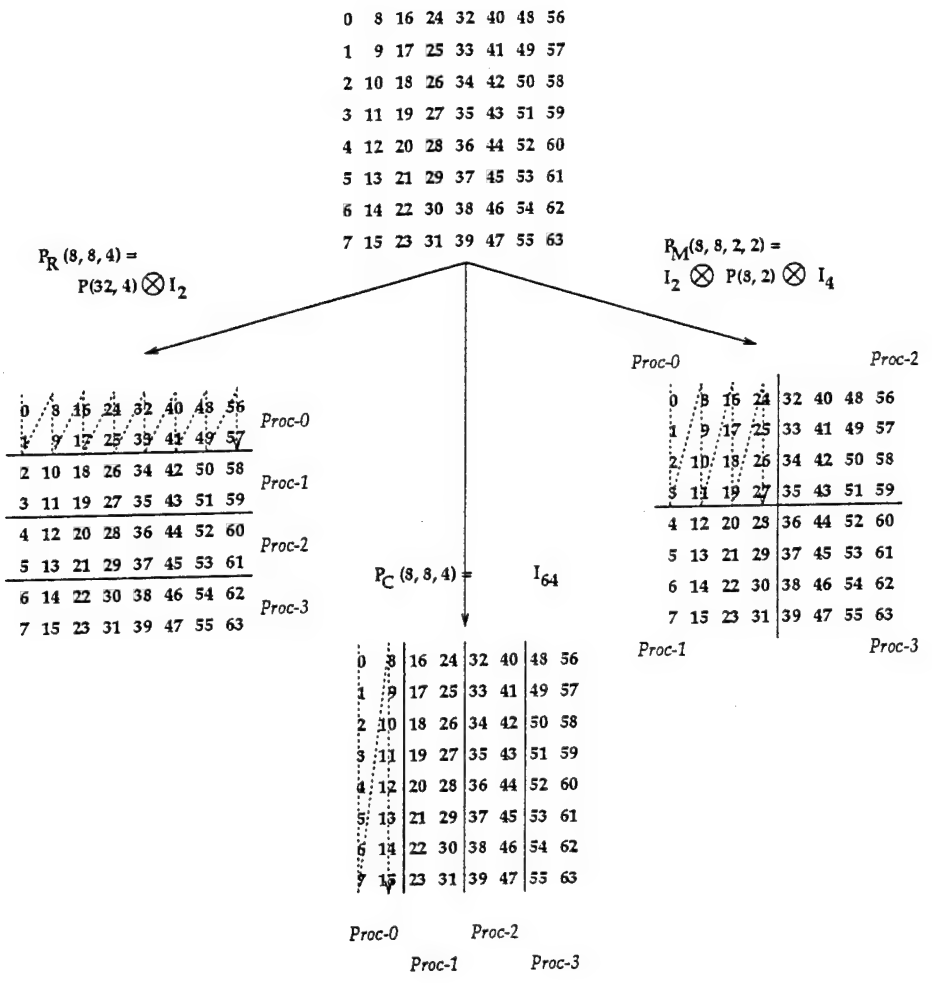


Figure 1: Action of data-partition algebraic expressions onto a 4-processor machine

desired data-partition of $\mathbf{a}$ among the processors where $\mathbf{Q}_1$ is one of the data-partition schemes ($\mathbf{P}_R$, $\mathbf{P}_C$, or $\mathbf{P}_M$) defined above. If one expects the output data to be in a particular partition after the computation, then resultant data is of the form $\hat{\mathbf{b}}$ where $\hat{\mathbf{b}} = \mathbf{Q}_2\mathbf{b}$ and $\mathbf{Q}_2$ is also one of the definitions $\mathbf{P}_R$, $\mathbf{P}_C$, or $\mathbf{P}_M$ defined above. Then parallel implementation corresponding to equation (21) after incorporating our definitions can be rewritten as:

$$\hat{\mathbf{b}} = \mathbf{Q}_2\ \mathbf{b} = \mathbf{Q}_2\ \mathbf{G}\ \mathbf{a} = \left[\mathbf{Q}_2\ \mathbf{G}\ \mathbf{Q}_1^{-1}\right]\hat{\mathbf{a}} \stackrel{\text{def}}{=} \hat{\mathbf{G}}\ \hat{\mathbf{a}} \tag{22}$$

Therefore, $\hat{\mathbf{G}} = \mathbf{Q}_2\ \mathbf{G}\ \mathbf{Q}_1^{-1}$ is the actual-operational matrix that takes into account of the complexity associated with the considered data-partition.

A.3.2 Moving Data among Distributed Memories

Once input data is partitioned among the processors, data migrations at the interfaces between individual algorithms may be necessary in order to achieve global optimal performance of an application. One frequently used data migration in numerical applications is well known matrix transpose. Let $\mathbf{a} = Vect_{MN}(\mathbf{A}_{M\times N})$, and $\mathbf{b} = Vect_{NM}(\mathbf{B}_{N\times M})$, where $\mathbf{B}_{N\times M}$ is the transpose of $\mathbf{A}_{M\times N}$. Then,

$$\mathbf{b}\ =\ P(MN, M)\ \mathbf{a}. \tag{23}$$

Hence $P(MN, M)$ is the operational matrix for transpose algorithms, that is, $\mathbf{G} = P(MN, M)$. When data-partition schemes are to be incorporated, the actual-operational matrix becomes $\hat{\mathbf{G}}$ (see equation (22)). That is,

$$P(MN, M) = \mathbf{Q}_2^{-1}\ \hat{\mathbf{G}}\ \mathbf{Q}_1, \tag{24}$$

and the equation (23) becomes

$$\hat{\mathbf{b}}\ =\ \hat{\mathbf{G}}\ \hat{\mathbf{a}}, \tag{25}$$

where $\hat{\mathbf{G}} = \mathbf{Q}_2\ P(MN, M)\ \mathbf{Q}_1^{-1}$. In the following, we present derivations for the operational matrices, $\hat{\mathbf{G}}$, required to transpose a matrix for the cases of row-, column-, and mesh-division data-partitions defined in previous subsection (assume $\mathbf{Q}_1 = \mathbf{Q}_2$ for simplicity), and visualize their implementation aspects from their tensor product formulations.

<u>Row-Division</u>

For row-division data-partition, we have

$$\hat{\mathbf{G}} = \mathbf{P}_R(N, M, k)\ P(MN, M)\ \mathbf{P}_R^{-1}(M, N, k). \tag{26}$$

According to definition (A.1), we have

$$\hat{\mathbf{G}} = \left[P(Mk,k) \otimes \mathbf{I}_{N/k}\right] P(MN,M) \left[P(Nk,N) \otimes \mathbf{I}_{M/k}\right], \tag{27}$$

(or)

$$\mathbf{G} = P(MN,M) = \left[P(Mk,M) \otimes \mathbf{I}_{N/k}\right] \hat{\mathbf{G}} \left[P(Nk,k) \otimes \mathbf{I}_{M/k}\right]. \tag{28}$$

Then, we can obtain expression for $\hat{\mathbf{G}}$ by rewriting $\mathbf{G} = P(MN,M)$ as:

$$P(MN,M) = \left[P(Mk,M) \otimes \mathbf{I}_{N/k}\right] \left[\mathbf{I}_k \otimes P(MN/k,M)\right]$$

by theorem (A.6)

$$P(MN,M) = \mathbf{P}_R^{-1}(N,M,k) \left[\mathbf{I}_{k^2} \otimes P(MN/k^2, M/k)\right] \left[(\mathbf{I}_k \otimes P(N,k)) \otimes \mathbf{I}_{M/k}\right]$$

by theorem (A.7) and equation (16)

$$P(MN,M) = \mathbf{P}_R^{-1}(N,M,k) \left[\mathbf{I}_k \otimes \mathbf{I}_k \otimes P(MN/k^2, M/k)\right] \left[P(k^2,k) \otimes \mathbf{I}_{N/k} \otimes \mathbf{I}_{M/k}\right] \left[P(Nk,k) \otimes \mathbf{I}_{M/k}\right]$$

by applying theorem (A.6) to $P(Nk,k)$

$$P(MN,M) = \mathbf{P}_R^{-1}(N,M,k) \left[\mathbf{I}_k \otimes \mathbf{I}_k \otimes P(MN/k^2, M/k)\right] \left[P(k^2,k) \otimes \mathbf{I}_{MN/k^2}\right] \mathbf{P}_R(M,N,k) \tag{29}$$

by definition (A.1)

$$P(MN,M) = \mathbf{P}_R^{-1} \hat{\mathbf{G}} \mathbf{P}_R.$$

Therefore, the actual-operational matrix in equation (25) for row-division partition can be expressed as two stages:

$$\hat{\mathbf{G}} = \left[\mathbf{I}_k \otimes \mathbf{I}_k \otimes P(MN/k^2, M/k)\right] \left[P(k^2,k) \otimes \mathbf{I}_{MN/k^2}\right]. \tag{30}$$

The first stage, $\underline{P(k^2,k) \otimes \mathbf{I}_{MN/k^2}}$, is a *global-task* involving message-passing among processors, since the expression does not contain an identity matrix, $\mathbf{I}_k$, on its left-hand side. The size of each message being passed is (MN/k^2) which is $(1/k)$th of the size of the data set residing at a processor. This is reflected in the above tensor product expression by $\mathbf{I}_{MN/k^2}$. The factor $P(k^2,k)$ in the expression indicates that each processor has $(k-1)$ subblocks to send out. Such message passing is carried out in $(k-1)$ stages with one subblock being kept within a processor. When the number of processors, k, is a 2-power integer, one-to-one communication structure can be obtained with **xor** binary operator and a pseudo-code implementation for this stage is shown in Table 1.

me = *my node number*
for **index** = 1 to $k-1$
myswap = xor(**me**,**index**)
Send block-**myswap** of my associated vector **a** to processor-**myswap**
Receive message from processor-**myswap**
Store message at block-**myswap** of my associated vector **a**
end

Table 1: Psuedo-code for message passing in transpose algorithms either for row-division or column-division partitions

The second stage, $\underline{\mathbf{I}_k \otimes \mathbf{I}_k \otimes P(MN/k^2, M/k)}$, represents a *local-task* due to the identity matrix $\mathbf{I}_k$ on its left-hand side. Each processor performs the parallel-stride operation $[\mathbf{I}_k \otimes P(MN/k^2, M/k)]$ locally.

Column-Division

For column-division data-partition, we have

$$\hat{\mathbf{G}} = \mathbf{P}_C(N, M, k)\ P(MN, M)\ \mathbf{P}_C^{-1}(M, N, k) \tag{31}$$

According to definition (A.2), we have

$$\hat{\mathbf{G}} = \mathbf{I}_{MN}\ \ P(MN, M)\ \ \mathbf{I}_{MN} = P(MN, M) = \mathbf{G}. \tag{32}$$

Then, we can obtain expression for $\hat{\mathbf{G}}$ as:

$$\begin{aligned} P(MN, M) &= [\mathbf{I}_k \otimes P(MN/k, M/k)] \left[P(Nk, k) \otimes \mathbf{I}_{M/k}\right] \\ &\text{by theorem (A.6)} \\ P(MN, M) &= [\mathbf{I}_k \otimes P(MN/k, M/k)] \left[\left\{\left(P(k^2, k) \otimes \mathbf{I}_{N/k}\right)(\mathbf{I}_k \otimes P(N, k))\right\} \otimes \mathbf{I}_{M/k}\right] \\ &\text{by theorem (A.7)} \\ P(MN, M) &= [\mathbf{I}_k \otimes P(MN/k, M/k)] \left[P(k^2, k) \otimes \mathbf{I}_{MN/k^2}\right] \\ &\quad \left[\mathbf{I}_k \otimes P(N, k) \otimes \mathbf{I}_{M/k}\right]. \end{aligned} \tag{33}$$

Therefore, the actual-operational matrix in equation (25) for column-division partitioning can be expressed as three stages:

$$\hat{\mathbf{G}} = [\mathbf{I}_k \otimes P(MN/k, M/k)] \left[P(k^2, k) \otimes \mathbf{I}_{MN/k^2}\right] \left[\mathbf{I}_k \otimes P(N, k) \otimes \mathbf{I}_{M/k}\right] \tag{34}$$

The first stage, $\underline{\mathbf{I}_k \otimes P(N,k) \otimes \mathbf{I}_{M/k}}$, represents *local* data permutations without message-passing due to the prior identity $\mathbf{I}_k$. Each processor performs the vector-stride operation $[P(N,k) \otimes \mathbf{I}_{M/k}]$ which moves N vectors with stride k, each vector is of length (M/k).

The second stage, $\underline{P(k^2,k) \otimes \mathbf{I}_{MN/k^2}}$, is a *global-task* that is similar to message-passing stage explained in row-division transpose algorithm. Hence the total communication is again $(k-1)$ messages, each message is of length (MN/k^2).

The final stage, $\underline{\mathbf{I}_k \otimes P(MN/k, M/k)}$, is a and simple-stride permutation stage with stride (M/k) *local* to each processor. All processors carry out the same operation in parallel without communication.

Mesh-Division

For mesh-division partition, we have

$$\hat{\mathbf{G}} = \mathbf{P}_M(N,M,k_2,k_1) P(MN,M) \mathbf{P}_M^{-1}(M,N,k_1,k_2). \tag{35}$$

According to definition (A.3) we have

$$\hat{\mathbf{G}} = \left[\mathbf{I}_{k_1} \otimes P(Mk_2/k_1, k_2) \otimes \mathbf{I}_{N/k_2}\right] P(MN,M) \left[\mathbf{I}_{k_2} \otimes P(Nk_1/k_2, N/k_2) \otimes \mathbf{I}_{M/k_1}\right], \tag{36}$$

(or)

$$P(MN,M) = \left[\mathbf{I}_{k_1} \otimes P(Mk_2/k_1, M/k_1) \otimes \mathbf{I}_{N/k_2}\right] \hat{\mathbf{G}} \left[\mathbf{I}_{k_2} \otimes P(Nk_1/k_2, k_1) \otimes \mathbf{I}_{M/k_1}\right]. \tag{37}$$

Then we can obtain expression for $\hat{\mathbf{G}}$ by decomposing $\mathbf{G} = P(MN,M)$ as follows.

$$\begin{aligned}
P(MN,M) &= [\mathbf{I}_{k_1} \otimes P(MN/k_1, M/k_1)] \left[P(Nk_1,k_1) \otimes \mathbf{I}_{M/k_1}\right] \\
&\text{by theorem (A.6)} \\
P(MN,M) &= \left[\mathbf{I}_{k_1} \otimes P(Mk_2/k_1, M/k_1) \otimes \mathbf{I}_{N/k_2}\right] [\mathbf{I}_k \otimes P(MN/k, M/k_1)] \\
&\quad \left[P(k,k_1) \otimes \mathbf{I}_{MN/k}\right] \left[\mathbf{I}_{k_2} \otimes P(Nk_1/k_2, k_1) \otimes \mathbf{I}_{M/k_1}\right] \qquad (38) \\
&\text{by theorem (A.7) on } P(MN/k_1, M/k_1) \text{ and by theorem (A.6) on } P(Nk_1,k_1) \\
P(MN,M) &= \mathbf{P}_M^{-1}(N,M,k_2,k_1) [\mathbf{I}_k \otimes P(MN/k, M/k_1)] \\
&\quad \left[P(k,k_1) \otimes \mathbf{I}_{MN/k}\right] \mathbf{P}_M(M,N,k_1,k_2) \qquad (39) \\
&\text{by equation (18) and definition (A.3)} \\
P(MN,M) &= \mathbf{P}_M^{-1}(N,M,k_2,k_1) \left[P(k,k_1) \otimes \mathbf{I}_{MN/k}\right]
\end{aligned}$$

$$[\mathbf{I}_k \otimes P(MN/k, M/k_1)]\, \mathbf{P}_M(M, N, k_1, k_2) \tag{40}$$

by commutative law

$$P(MN, M) \;=\; \mathbf{P}_M^{-1}\, \hat{\mathbf{G}}\, \mathbf{P}_M$$

Therefore, the actual-operational matrix in equation (25) for mesh-division partition can be expressed as two stages in two different ways (equations (39) and (40)):

(a) $\hat{\mathbf{G}} = [\mathbf{I}_k \otimes P(MN/k), M/k_1)] \left[P(k, k_1) \otimes \mathbf{I}_{MN/k}\right]$, and

(b) $\hat{\mathbf{G}} = \left[P(k, k_1) \otimes \mathbf{I}_{MN/k}\right] [\mathbf{I}_k \otimes P(MN/k, M/k_1)]$.

In case of (a), the first stage, $\underline{P(k, k_1) \otimes \mathbf{I}_{MN/k}}$, is a *global-task* involving message-passing since there is no prior identity matrix. In fact, it is a single message-passing routine with message size being (MN/k) as compared to $(k-1)$ messages each of size (MN/k^2) in either row-division or column-division transpose algorithms.

The second stage, $\underline{\mathbf{I}_k \otimes P(MN/k, M/k_1)}$, represents that each processor executes a *local* simple-stride permutation because of the prior identity matrix $\mathbf{I}_k$. In fact, if we consider data at each processor to be a matrix of size $M/k_1 \times N/k_2$, then action to be performed in this stage is k local matrix transposes that are performed simultaneously on k processors.

A.3.3 Measured Timing of the Three Transpose Algorithms

Transpose algorithms derived in Section A.3.2 are implemented on Intel's Paragon. The measured execution times of the three transpose algorithms are tabulated in Table 2. From the derivations in equations (30), (33), (39), and (40), we have seen that transposing a matrix of size $M \times N$ on a k-processor machine for row- and column-division each requires $(k-1)$ communications with the size of each message being (MN/k^2). For mesh-division *one* communication of size (MN/k) is needed. Though message length in mesh-division is k times more than that of any message in either row-division or column-division, results in Table 2 show that transpose algorithm for mesh-division reduces the overheads to initiate communications. Smaller number of long messages can take advantages of the pipelined nature of wormhole routing [15]. These results also show that unlike uniprocessor algorithms, variations in data-decompositions can have a great impact on the performance of an algorithm.

M	N	Row-Division (msec)	Col-Division (msec)	Mesh-Division (msec)
128	128	5.236	6.172	1.316
128	256	5.902	7.051	2.028
128	512	9.031	10.409	2.159
128	1024	12.356	15.312	3.866
256	128	5.501	6.665	1.825
256	256	8.283	9.746	2.301
256	512	11.483	14.027	4.018
256	1024	20.076	22.503	7.548
512	128	8.310	9.432	3.450
512	256	11.555	13.359	5.905
512	512	18.536	21.122	7.954
512	1024	39.628	38.529	16.434
1024	128	11.228	13.132	5.815
1024	256	17.526	20.616	10.631
1024	512	31.211	37.445	20.889
1024	1024	50.936	66.403	49.274

Table 2: Experimental results of transpose algorithms on 8-node Intel's Paragon. Explanation: Transpose algorithms for Row-division and Column-Division require seven small communications while that in mesh requires only one large communication. Effect of communication overhead on transpose algorithm clearly results mesh-division more efficient than the other cases. Among row-division and column-division structures, row-division requires only one local permutation while column-division requires one local permutation before the communications and another after that. This is also seen from third and fourth columns.

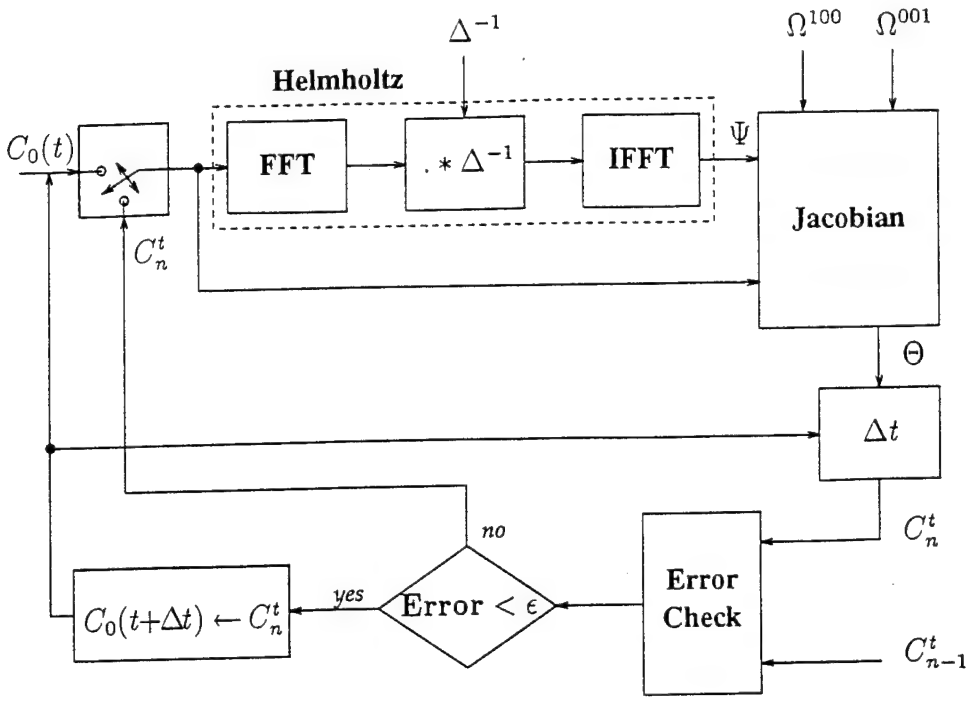


Figure 2: Flow Chart for computation of coefficients of Vorticity

A.4 Application Examples

A.4.1 An Application in Fluid Mechanics

This application solves Euler partial differential equation using wavelet-Galerkin method [12, 16]. Figure 2 shows the flowchart for evaluating the coefficients of vorticity in fluid mechanics at each time-step. The major computation blocks in the figure are *Jacobian* and *Helmholtz* while other modules such as *Error Check,* computation of vorticity coefficients in next step (Δt) are not time consuming. It is well known that Jacobian prefers mesh-division data-partitioning because of boundary conditions, that is, data dependency exists along the four edges of a grid. However, Intel's distributed memory machines have efficient two dimensional fast Fourier transform algorithms (2D-FFT) based on row- or column-division data-partitions. Therefore, switching between different data-partition schemes is necessary to carry out the computation of this application efficiently. First, we need to convert the mesh-division data-partitioning to row- or column-division at the output of Jacobian (the input of Helmholtz). Then, we need to convert back to mesh-division at the input of the Jacobian (output of Helmholtz).

Converting data-partitioning schemes at the interfaces of different computational modules can be very expensive since it involves massive amount of data movements. The communication cost

caused by such data movements may well dominate the total computation cost of applications even though individual algorithms are optimized. With our formal definitions of data-partitions, however, manipulation of communication cost become straightforward.

Let us consider the computation of the Helmholtz on a k-processor distributed memory system. Assume that k can be factored as $k = k_s \times k_s$. The input data to this computation module is in mesh-division, which can be represented by $\mathbf{P}_M$. With this input format, we perform 2D-FFT and its inverse (2D-IFFT). The summation form of the 2D-DFT on matrix $\mathbf{X}$ of size $M \times N$ is given by:

$$\mathbf{Y}(k,l) = \sum_{m=0}^{M-1} \left[\sum_{n=0}^{N-1} \mathbf{X}(m,n) e^{-j\frac{2\pi nl}{N}} \right] e^{-j\frac{2\pi mk}{M}}. \tag{41}$$

The tensor products representation of equation (41) can be written as:

$$\mathbf{y} = \underbrace{[\mathbf{F}_N \otimes \mathbf{F}_M]}_{\mathbf{G}} \mathbf{x}, \tag{42}$$

where $\mathbf{F}_J$ is a $J \times J$ matrix with entries $F(i,k) = \exp(-j2\pi ik/J)$, $j = \sqrt{-1}$, $\mathbf{y} = Vect_{MN}(\mathbf{Y})$, $\mathbf{x} = Vect_{MN}(\mathbf{X})$, and $\mathbf{G}$ is the operational matrix.

To compute equation (42) on a k-processor parallel machine, we first parallelize the operational matrix by inserting identity matrices under the assumption that k divides both M and N. There are two ways of decomposing the equation (42): (a) $\mathbf{y} = [\mathbf{I}_N \otimes \mathbf{F}_M][\mathbf{F}_N \otimes \mathbf{I}_M]\ \mathbf{x}$, which first computes Fourier transforms on columns (using one dimensional FFT routines) followed by transforms on rows, and (b) $\mathbf{y} = [\mathbf{F}_N \otimes \mathbf{I}_M][\mathbf{I}_N \otimes \mathbf{F}_M]\ \mathbf{x}$, which performs transformation on rows followed by that on columns. These two decompositions are well known as *row-column* decomposition for transform methods. Consider the first decomposition (a). The factor on the left-hand side represents a parallel computation of $\mathbf{F}_M$ because of the preceding identity matrix $\mathbf{I}_N$ while the factor on the right-hand side cannot be done in parallel. To parallelize this stage of computation, we apply the commutative law presented in theorem (A.2), resulting in

$$\mathbf{y} = [\mathbf{I}_N \otimes \mathbf{F}_M]\ P(MN,N)\ [\mathbf{I}_M \otimes \mathbf{F}_N]\ P(MN,M)\ \mathbf{x} \tag{43}$$

If it is required that the Fourier transformed data be in the same data-partition scheme as the original data (say mesh-division), then input matrix is $\hat{\mathbf{x}} = \mathbf{P}_M\ \mathbf{x}$ and output matrix is $\hat{\mathbf{y}} = \mathbf{P}_M\ \mathbf{y}$ ($\mathbf{P}_M = \mathbf{P}_M(M,N,k_s,k_s) = \mathbf{I}_{k_s} \otimes P(N,N/k_s) \otimes I_{M/k_s}$). Equation (43) can be rewritten as:

$$\hat{\mathbf{y}} = \mathbf{P}_M\ [\mathbf{I}_N \otimes \mathbf{F}_M]\ P(MN,N)\ [\mathbf{I}_M \otimes \mathbf{F}_N]\ [P(MN,M)\ \mathbf{P}_M^{-1}]\ \hat{\mathbf{x}} \tag{44}$$

If we use the second parallelization, (b), we have

$$\hat{\mathbf{y}} = [\mathbf{P}_M P(MN, N)]\,[\mathbf{I}_M \otimes \mathbf{F}_N]\,P(MN, M)\,[\mathbf{I}_N \otimes \mathbf{F}_M]\,\mathbf{P}_M^{-1}\hat{\mathbf{x}} \tag{45}$$

In the following, we will see how we utilize our new definitions on data-partition and migration to maximize the parallelism and minimize the communication cost while computing equation (45). From equation (43), we can see that two transpose algorithms are required $P(MN, N)$ and $P(MN, M)$. Each of these transpose algorithms needs $(k-1)$ stages of message-passing on a k-processor machine as evidenced in the last subsection. We will show in the following how we reduce the communication cost of one of the two transpose algorithms from $(k-1)$ to $(2k_s - 2)$ by manipulating the algorithm expressions. Note that $k = k_s^2$.
Now, let us consider equation (45). The first stage of computation is $\mathbf{P}_M^{-1}$ which converts the mesh-division into column-division for FFT computation ($\mathbf{P}_C\ \mathbf{P}_M^{-1} = \mathbf{P}_M^{-1}$ since $\mathbf{P}_C$ is an identity matrix). According to our definition, we have

$$\mathbf{P}_M^{-1} = \left[\mathbf{I}_{k_s} \otimes P(N, N/k_s) \otimes \mathbf{I}_{M/k_s}\right].$$

Using equation (18) and theorem (A.6), we have

$$\mathbf{P}_M^{-1} = \underbrace{\left[\mathbf{I}_k \otimes P(N/k_s, N/k) \otimes \mathbf{I}_{M/k_s}\right]}_{\mathcal{Z}_2} \underbrace{\left[\mathbf{I}_{k_s} \otimes P(k, k_s) \otimes \mathbf{I}_{MN/k_s^3}\right]}_{\mathcal{Z}_1} \tag{46}$$

The above factorization on $\mathbf{P}_M^{-1}$ results in two stages. The first stage, $\mathcal{Z}_1$, involves $(k_s - 1)$ communications with k_s columns of processors communicating in parallel. The second stage, $\mathcal{Z}_2$, is a local vector-stride data-shuffling.

Similarly, at the output of the FFT, we can also manipulate the algebraic expression for mesh-division data-partitioning. The last stage of equation (45) converts back to mesh-division data-partition, which can be simplified as follows.

$$\begin{aligned}
&\mathbf{P}_M P(MN, N) \\
&\quad = \left[\mathbf{I}_{k_s} \otimes P(N, k_s) \otimes \mathbf{I}_{M/k_s}\right]\left[P(Nk_s, N) \otimes \mathbf{I}_{M/k_s}\right] \\
&\qquad [\mathbf{I}_{k_s} \otimes P(MN/k_s, N)] \\
&\text{by theorem (A.7) and definition (A.3)} \\
&\quad = \underbrace{\left[P(k, k_s) \otimes \mathbf{I}_{MN/k}\right]}_{\mathcal{Z}_{11}} [\mathbf{I}_{k_s} \otimes P(MN/k_s, N)]
\end{aligned}$$

$$
\begin{aligned}
&= \mathcal{Z}_{11}\left[\mathbf{I}_{k_s} \otimes P(Nk_s, N) \otimes \mathbf{I}_{M/k}\right] \underbrace{\left[\mathbf{I}_k \otimes P(MN/k, N)\right]}_{\mathcal{Z}_8} \\
&= \mathcal{Z}_{11} \underbrace{\left[I_k \otimes P(N, N/k_s) \otimes \mathbf{I}_{M/k}\right]}_{\mathcal{Z}_{10}} \underbrace{\left[\mathbf{I}_{k_s} \otimes P(k, k_s) \otimes \mathbf{I}_{MN/k_s^3}\right]}_{\mathcal{Z}_9} \mathcal{Z}_8
\end{aligned} \tag{47}
$$

From the transpose algorithm derived for mesh-division partition in Section A.3.2, we know that $\mathcal{Z}_{11}$ represents *one* single communication. Stage $\mathcal{Z}_9$ is also a transpose algorithm similar to stage $\mathcal{Z}_1$ that requires $(k_s - 1)$ communications. Therefore, the total number of communications required to carry out the FFT is $(k + 2k_s - 2)$ as compared to $(2k - 2)$ for direct interfacing between the Helmholtz and Jacobian.

Another variant of the computation can also be obtained easily by manipulating the tensor algebra in a different way. Consider the last stage, $[\mathbf{P}_M P(MN, N)]$. First, we factor $P(MN, N)$ as follows.

$$
\begin{aligned}
P(MN, N) &= \left[\mathbf{I}_{k_s} \otimes P(MN/k_s, N/k_s)\right]\left[P(Mk_s, k_s) \otimes \mathbf{I}_{N/k_s}\right] \\
&\quad \text{by theorem (A.6)} \\
&= \left[\mathbf{I}_{k_s} \otimes \left(P(N, N/k_s) \otimes \mathbf{I}_{M/k_s}\right)\left(\mathbf{I}_{k_s} \otimes P(MN/k, N/k_s)\right)\right] \\
&\quad \left[P(Mk_s, k_s) \otimes \mathbf{I}_{N/k_s}\right] \\
&\quad \text{by theorem (A.7) on } P(MN/k_s, N/k_s) \\
&= \mathbf{P}_M^{-1}\left[\mathbf{I}_k \otimes P(MN/k, N/k_s)\right]\left[P(Mk_s, k_s) \otimes \mathbf{I}_{N/k_s}\right]. \\
&\quad \text{by equation (18)}
\end{aligned} \tag{48}
$$

Therefore by theorem (A.7) we have,

$$
\begin{aligned}
\mathbf{P}_M P(MN, N) &= \left[\mathbf{I}_k \otimes P(MN/k, N/k_s)\right]\left[P(Mk_s, k_s) \otimes \mathbf{I}_{N/k_s}\right] \\
&= \underbrace{\left[\mathbf{I}_k \otimes P(MN/k, N/k_s)\right]}_{\mathcal{Z}_{10}} \underbrace{\left[P(k_s^3, k_s) \otimes \mathbf{I}_{MN/k_s^3}\right]}_{\mathcal{Z}_9} \\
&\quad \underbrace{\left[\mathbf{I}_k \otimes P(M/k_s, k_s) \otimes \mathbf{I}_{N/k_s}\right]}_{\mathcal{Z}_8}
\end{aligned} \tag{49}
$$

Once again we reduced total communication cost from $(2k - 1)$ to $(k + 2k_s - 3)$, eliminating the *one* large and final communication from the previous variant.

Experiments of running the complete application based on our derivation above have been carried out on Intel's iPSC/860. The execution times of the important computational modules

Nodes	Jacobian		Helmholtz			Total		
	row-D	Mesh	row-D	Mesh1	Mesh2	row-D	Mesh1	Mesh2
4	2.8317	2.7939	0.11216	0.18218	0.16298	2.9438	2.9761	2.9568
16	0.8128	0.7310	0.06094	0.09950	0.07688	0.8738	0.8305	0.8079
64	0.3095	0.1996	0.10510	0.12022	0.08916	0.4146	0.3198	0.2887

Table 3: Timing results for 128×128 size vorticity computations Explanation: Results demonstrate that by restructuring at the interface of Jacobian and Helmholtz using our data-partition expressions, we could improve the efficiency of Jacobian at the cost of a slight decrease in efficiency of Helmholtz. This resulted in total improvement of the efficiency of application.

as well as the total execution time were measured. The results reported in Table 3 are averaged over a hundred runs. The columns marked row-D are the execution times of row-division while those marked Mesh1 and Mesh2 are for two variants of mesh-division computation derived above. From this table, performance improvement of up to 43.61% is observed.

A.4.2 A New FFT Algorithm

Existing machine library on Intel's multiprocessors for FFT computation are based on row-division or column-division. From our new definitions of data-partitions, we developed a new communication structures for parallel FFT algorithm [17]. The main idea is to partition data according to mesh-division. Rewriting equation (45), we have

$$\hat{\mathbf{y}} = \mathbf{P}_M \; P(MN, N) \; [\mathbf{I}_M \otimes \mathbf{F}_N] \; P(MN, M) \; [\mathbf{I}_N \otimes \mathbf{F}_M] \mathbf{P}_M^{-1} \hat{\mathbf{x}}, \tag{50}$$

We have seen transpose algorithms similar to the one in the above equation for row- or column-division FFT algorithms. Each transpose algorithm requires $(2k-1)$ communications. Now, use the following equality (see equation (40)) to substitute $P(MN, M)$ in the above equation.

$$P(MN, M) = \mathbf{P}_M^{-1} \left[P(k, k_1) \otimes \mathbf{I}_{MN/k} \right] [\mathbf{I}_k \otimes P(MN/k, M/k_1)] \, \mathbf{P}_M \tag{51}$$

Similarly, for $P(MN, N)$, we interchange the roles of M and N, and k_1 and k_2 in equation (39). We have

$$P(MN, N) = \mathbf{P}_M^{-1} \, [\mathbf{I}_k \otimes P(MN/k, N/k_2)] \left[P(k, k_2) \otimes \mathbf{I}_{MN/k} \right] \mathbf{P}_M.$$

Substitute the above equality to equation (50). Then, the 2D-FFT algorithm becomes

$$\hat{\mathbf{y}} \;\; = \;\; [\mathbf{I}_k \otimes P(MN/k, N/k_2)] \left[P(k, k_2) \otimes \mathbf{I}_{MN/k} \right]$$

$$\left[\mathbf{P}_M\left(\mathbf{I}_M \otimes \mathbf{F}_N\right)\mathbf{P}_M^{-1}\right]\left[P(k,k_1)\otimes \mathbf{I}_{MN/k}\right]$$
$$\left[\mathbf{I}_k \otimes P(MN/k, M/k_1)\right]$$
$$\left[\mathbf{P}_M\left(\mathbf{I}_N \otimes \mathbf{F}_M\right)\mathbf{P}_M^{-1}\right]$$

Note that terms $[P(k,k_2)\otimes \mathbf{I}_{MN/k}]$ and $[P(k,k_1)\otimes \mathbf{I}_{MN/k}]$ are dummy operations with respect to implementation because these permutations represent exchange of entire data residing at different nodes. This can be done by addressing processors according to the required permutation instead of data movement. Operations that start with $\mathbf{I}_k$ are parallel operations with no communication. For the remaining two terms that involve $\mathbf{P}_M$ and $\mathbf{P}_M^{-1}$, we can decompose $\mathbf{P}_M$ and $\mathbf{P}_M^{-1}$ as following

$$\begin{aligned}\mathbf{P}_M &= \left[\mathbf{I}_{k_2}\otimes P(k_1^2,k_1)\otimes \mathbf{I}_{MN/k_1^2k_2}\right]\left[\mathbf{I}_k \otimes P(N/k_2,k_1)\otimes \mathbf{I}_{M/k_1}\right]\\ \mathbf{P}_M^{-1} &= \left[\mathbf{I}_k \otimes P(N/k_2,N/k)\otimes \mathbf{I}_{M/k_1}\right]\left[\mathbf{I}_{k_2}\otimes P(k_1^2,k_1)\otimes \mathbf{I}_{MN/k_1^2k_2}\right]\end{aligned}$$

Each of $\mathbf{P}_M$ and $\mathbf{P}_M^{-1}$ has one communication stage and one local permutation stage. Each of these communication stages transmits (k_1-1) messages, with the size of each message being $MN/k_1^2k_2$. For the other dimension, each of the $\mathbf{P}_M$ and $\mathbf{P}_M^{-1}$ will have (k_2-1) communications with size of each message being $(MN/k_1k_2^2)$. Therefore, the total number of communications is reduced from $O(k_1 * k_2)$ to $O(k_1+k_2)$.

Experiments to measure the actual performance of the above 2D-FFT and the existing library routine on the Touchstone Delta machine have been carried out. The measurements are reported in Table A.4.2. The results shown in this table are measured with a library routine called `dclock()` that returns a double precision number. Using this routine at the beginning and at the end of each of the algorithms, we obtained double precision time in milliseconds. These timings are purely for execution of the task because processors are not time-sharing by multiple users. However, since each node would execute in a slightly different time due to the underlying asynchronous communication network of machines, we considered the maximum value of the times reported by all the nodes. Also, we have averaged timings over a set of one hundred experiments with forward and inverse two-dimensional transforms for each data size.

Performance of two different implementations are reported by executing them on 128-node and 256-node machine. Various data sizes that we have tested are presented in the first column in the table. Second and third columns represent timings for existing and new approaches, respectively, on 128-node machine while fourth and fifth columns are for the cases of 256-node

Dimensions			128 nodes		256 nodes	
			Old	New	Old	New
M	×	N	*(msecs)*	*(msecs)*	*(msecs)*	*(msecs)*
128	×	128	120.117	27.727	193.481	31.711
256	×	128	120.151	31.234	192.980	35.017
256	×	256	121.681	34.165	245.634	39.499
512	×	128	125.425	34.401	210.761	35.865
512	×	256	129.847	44.944	254.412	44.948
1024	×	128	128.236	44.883	227.441	43.225
512	×	512	125.901	60.946	270.365	56.096
1024	×	256	133.562	64.331	262.051	53.420
1024	×	512	152.919	99.989	285.066	76.041
1024	×	1024	211.274	177.306	294.038	119.288

Table 4: Results on Intel's *i860* based DELTA machine. Explanation: These results reflect the variations in communication structure for "new" and "old" algorithms because "new" algorithm requires 44 and 60 communications for the implementations on 128 (16×8) and 256 (16×16) processor systems, respectively, while "old" algorithm requires 254 and 510 communications, respectively. However, it is to be noted that reduction in number of communications in "new" algorithm is traded-off with size of the data begin communicated.

machine. It can be seen from the table that performance gains of the new FFT are significant. We observed up to 600% performance improvement over the existing machine library.

A.5 Related Work

Data organization is the key to successful parallelization of data parallel programs. As indicated in the introduction, there are two tracks of efforts in data-partition and migration in distributed memory multiprocessors: automatic data-partitioning for general loop constructs as part of compiler and optimal partitioning for a specific algorithm. In this subsection, we briefly summarize the existing works in this field as related to our work presented in this paper. For more comprehensive review of previous work in data-partitioning and redistribution, readers are referred to [3, 2, 5].

Ramanujam and Sadayappan [2] studied compile-time techniques for data-partitioning in distributed memory systems. They presented an analysis of communication-free partitions with a nice geometric demonstration. The research work performed by Li and Chen [6] focused on minimizing data movement among processors due to cross-references of multiple distributed arrays (alignment of multiple data structures). They have also presented a method of automatically generating efficient message-passing routines in parallel programs [6]. Gupta and Banerjee introduced the notion of constraints on data-partitioning to obtain good performance. In [9], a compiler algorithm was described to automatically finds optimal parallelism and optimal locality in general loop nesting. All these studies aimed at optimizing data-partition and data alignments as part of compiler. It is known that such optimization problem is NP-complete. A number of heuristics have been proposed [6, 7, 8, 2, 18, 1].

The use of tensor product notation to describe parallel algorithms has a long history beginning with Pease [19]. Johnson *et al* [20] presented a comprehensive discussion on how to use tensor notations to design, modify and implement FFT algorithms on various computer architectures. Attempts to derive variants of FFT algorithms keeping the underlying architecture in mind have proven successful [10, 13]. Huang, Johnson and Johnson [21] have recently used tensor notations for formulating Strassen's matrix multiplication algorithm. Using the tensor representation, they derived three variant programs and compared their performance characteristics for shared memory multiprocessors.

Kaushik, Huang, Johnson and Sadayappan have proposed a very nice approach for data redistribution in distributed memory systems, which appeared recently in [5]. While their approach also utilizes the tensor notation as a tool, our work differs in several aspects. First of all, our definitions are expressed in matrix forms while theirs are in terms of indices (tensor bases). With their model one can estimate communication cost of a computation precisely while with our formulations one can easily manipulate the communication structures of a computation to achieve optimal performance. Deriving variants of an algorithm using our definitions are relatively simple because the data communication is easily visible. Secondly, all the definitions presented in [5] such as *cyclic, block*, and *block cyclic* can be defined using our formulations as evidenced in Section 3, whereas some of data-partitions such as mesh-division cannot be easily expressed using the notations in [5]. In addition, our representation acts directly on data vector $\mathbf{a}(O : N-1)$ to achieve the required data-partition and migration scheme while their representation presents ways to manipulate data indices from one distribution to the

other (redistribution). Unlike their representation, we can embed our expressions for data distribution into an algorithm. As a result, global optimization of an application consisting of several computation modules become straightforward by just manipulating the algebraic expression at the interfaces between individual algorithms.

A.6 Conclusions

In this paper, we have presented a formal description for data-partition in distributed memory multiprocessors. Using the algebra of tensor products and stride permutations, different schemes of storing data in a distributed memory system are represented in a compact and systematic manner. The formalism of various data-partitioning schemes allows for immediate embedding of an algebraic expression into a computational algorithm. As a result, optimization of data-partition becomes simple tensor algebra manipulations. We have demonstrated the usefulness and significance of our formulations by considering applications. Experiments on existing distributed memory machines have been carried out. Numerical results show that significant performance gains are possible by using our formulations to generate variants of an algorithm tailoring to specific system architectures.

References

[1] H. Xu and L. M. Ni, "Optimizing Data Decomposition for Data Parallel Programs," in *International Conference on Parallel Processing*, pp. 225–232, 1994.

[2] J. Ramanujam and P. Sadayappan, "Compile-Time Techniques for Data Distribution in Distributed Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 472–482, Oct. 1991.

[3] M. Gupta and P. Banerjee, "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compiler on Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 179–193, Mar. 1992.

[4] S. K. S. Gupta, S. D. Kaushik, S. Mufti, S. Sharma, C. H. Huang, and P. Sadayappan, "On Compiling Array Expressions F Efficient Execution on Distributed-Memory Machines," in *Proceedings of International Conference on Parallel Processing*, pp. 301–305, 1993.

[5] S. D. Kaushik, C.-H. Huang, R. W. Johnson, and P. Sadayappan, "An Approach to Communication-Efficient Data Redistribution," in *Supercomputing 94*, pp. 364–373, 1994.

[6] J. Li and M. Chen, "The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines," *Journal of Parallel and Distributed Computing*, vol. 13, pp. 213–221, Oct. 1991.

[7] K. Knob, J. D. Lukas, and G. L. Steel, "Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines," *Journal of Parallel and Distributed Computing*, vol. 3, pp. 102–118, Feb. 1990.

[8] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S. H. Teng, "Automatic Array Alignment in Data Parallel Algorithms," in *Twentieth Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, pp. 16–28, Jan. 1993.

[9] J. M. Anderson and M. S. Lam, "Global Optimization for Parallelism and Locality on Scalable Parallel Machines," in *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation*, pp. 112–125, June 1993.

[10] M. An, I. Gertner, M. Rofheart, and R. Tolimieri, "Discrete Fast Fourier Transform Algorithms: A Tutorial Survey," *Advances in Elec. and Electron Physics*, vol. 80, 1991.

[11] G. Fox, A. J. C. Hey, and S. Otto, "Matrix Algorithms on the Hypercube I: Matrix Multiplication," *Parallel Computing*, vol. 4, pp. 17–31, 1987.

[12] Z. Qian and J. Weiss, "Wavelets and The Numerical Solution of Partial Differential Equations," *Journal of Computational Physics*, vol. 106, pp. 155–175, 1993.

[13] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag Publishing Company, 1989.

[14] M. Davio, "Kronecker Products and Shuffle Algebra," *IEEE Transactions on Computers*, vol. C-30, pp. 116–125, 1981.

[15] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, pp. 62–76, 1993.

[16] J. Weiss, "Wavelets and The Study of Two-Dimensional Turbulence," Technical Report AD910628, Aware Inc., One Memorial Dr., Cambridge, MA 02142-1301, 1992. and the Proceedings of French-USA Workshop on *Wavelets and Turbulence*, Princeton University, June 1991, Ed. Y. Maday, Springer-Verlag.

[17] N. Anupindi, M. An, J. W. Cooley, and Q. Yang, "A New and Efficient FFT Algorithm for Distributed Memory Systems," *to appear in International Conference on Parallel and Distributed Systems-94*, 1994.

[18] D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "IBM system/360 model 91: Machine philosophy and instruction handling," *IBM J. Research and Development*, pp. 8–24, Jan. 1967.

[19] M. C. Pease, "An adaptation of the fast Fourier transform for parallel processing," *J. ACM*, vol. 15, pp. 252–264, Apr. 1968.

[20] J. R. Johnson, R. W. Johnson, D. Rodriguez, and R. Tolimieri, "A Methodology for Designing, Modifying, and Implementing Fourier Transform Algorithms on Various Architectures," *IEEE Transactions on Circuits, Systems, and Signal Processing*, vol. 9, pp. 449–500, 1990.

[21] C.-H. Huang, J. R. Johnson, and R. W. Johnson, "A Tensor Product Fromulation of Strassen's Matrix Multiplication Algorithm," *App. Math. Lett.*, vol. 3, pp. 67–71, 1990.

B Efficient Multidimensional DFT Module Implementation on the INTEL i860 Processor

Abstract

In this paper, we present a unified implementation methodology for computing large one- and multi-dimensional Fourier transforms. By formulating various DFT algorithms in the language of tensor products, any large size Fourier transform is built up by a collection of small size DFT modules which include as parameters decimation step sizes and twiddle factors. These parameters are introduced in the DFT modules to take advantage of modern computer architectures with parallel, pipelined, multi-functional structures, while providing flexibility into the building blocks.

B.1 Introduction

Continuing our work [1] presented in ICSPAT'92, we have developed a unified implementation methodology for computing large one- and multi-dimensional Fourier transforms. Tensor product formulation of various DFT algorithms plays a central role in unifying implementation by identifying small number of computational cores and necessary parameters. Our library of core computation modules has the following features:

- We have efficiently implemented prime factors 3, 5, 7, 11, 13, 17 as well as powers of 2. Thus, transform size on each dimension of a multi-dimensional Fourier transform can have factors other than 2.
- One-dimensional small modules take advantage of vector operations on i860 by looping on other factors of the same dimension and other dimensions.
- One-dimensional small modules have pre-calculated twiddle factor array as a parameter. This provides for intermediate stages of Cooley-Tukey FFT implementation.

It is widely believed that data size on each dimension must be a power of two. In fact, a popular reference on numerical methods [2] recommends that if the data are defined over a period whose size is not a power of two, they are to be filled with zeros up to the next power of two. In multi-dimensional DFT computation, this will increase the transform size dramatically, not only slowing down the computation but also causing cache thrash and memory overflow. In the case of the parallel computer iPSC/860, each node processor has 8M byte memory. If

the size of complex data to be processed is $72 \times 72 \times 72 = 373,248$, computation is made in the local memory of the processing unit without data segmentation. On the other hand, by padding with zeros, the size of complex data to be processed will be $128 \times 128 \times 128 = 2,097,152$, which is beyond the capacity of local memory; segmentation and data loading in and out will cause severe problem.

In this paper, we will describe an implementation strategy for efficient multi-dimensional DFT routines on the Intel i860 processor. Timing results of some sample medium size of 2-dimensional DFT modules with prime factor on each dimension is provided. The results of comparable power of 2 FFT package [6] that are commercially avaiable are also included.

B.2 Tensor Product Formulation

The tensor product presentation of fast Fourier transform algorithms dates back to Pease's paper [7] of 1968. Its role in application has varied during this period, from that of a notational convenience for describing a complex algorithm to that of an interactive programming tool. A detailed discussion on tensor product identities can be found in [8]. In this paper, we emphasize the tensor product as a programming tool in DFT module implementation. The parameters that govern the data permutation, vector segmentation, an algorithm's granularity and parallelism, come naturally from tensor product formulation of various algorithms.

One-dimensional N-point Fourier transform of array $\underline{x}$ is defined as

$$\underline{y} = F(N)\underline{x}. \tag{1}$$

where $F(N)$ is an $N \times N$ matrix defined by

$$F(N) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{(N-1)} \\ \vdots & & & & \\ 1 & w^{(N-1)} & w^{2(N-1)} & \cdots & w^{(N-1)^2} \end{bmatrix}, \tag{2}$$

where $w = e^{-j2\pi/N}$.

The $N_1 \times N_2$ 2-dimensional Fourier transform of X, denoted by

$$F(N_1, N_2)X \tag{3}$$

can be written in a matrix form as

$$Y = F(N_1)XF(N_2), \tag{4}$$

where X and Y are $N_1 \times N_2$ 2-dimensional input and output arrays respectively.

Denote by $\underline{x}$ the vector in C^N, $N = N_1 N_2$, formed by reading in order, down the columns of X, and $\underline{y}$ formed the same way from Y. We can write the 2-dimensional Fourier transform in a tensor product format:

$$\underline{y} = (F(N_2) \otimes F(N_1))\underline{x}. \tag{5}$$

(5) can be factored as:

$$\underline{y} = (F(N_2) \otimes I_{N_1})(I_{N_2} \otimes F(N_1))\underline{x}. \tag{6}$$

(6) is usually refered to as the row-column method: $I_{N_2} \otimes F(N_1)$ computes on the rows, and $F(N_2) \otimes I_{N_1}$ computes on the columns.

The tensor product formulation of 2-dimensional Fourier transform in (5) provides a general format for multi-dimensional Fourier transforms. Denote the K-dimensional Fourier transform of array X of size $N_1 \times N_2 \times \cdots \times N_K$ is denoted by

$$Y = F(N_1, N_2, \cdots, N_K)X \tag{7}$$

Denote by $\underline{x}$ the vector in C^N, $N = N_1 N_2 \cdots N_K$, formed by reading in order down the columns of X along N_1 dimension and then N_2 till N_K dimension, and $\underline{y}$ formed the same way from Y, we can write multi-dimensional Fourier transform of (7) in a tensor product format:

$$\underline{y} = (F(N_K) \otimes \cdots \otimes F(N_2) \otimes F(N_1))\underline{x}. \tag{8}$$

(8) can be factorized into K stages of Fourier transform computation.

$$\underline{y} = (F(N_K) \otimes I_{N_{K-1} \cdots N_1}) \cdots$$
$$(I_{N_K \cdots N_3} \otimes F(N_2) \otimes I_{N_1})(I_{N_K \cdots N_2} \otimes F(N_1))\underline{x}. \tag{9}$$

Every stage of (9) is of the form

$$I_L \otimes F(M) \otimes I_S. \tag{10}$$

The structure of (10) suggests a unified implementation methodology of multi-dimensional Fourier transform by a set one-dimensional DFT modules with parameters L and S: S determines the stride permutation; L determines the looping.

The tensor product formulation of multi-dimensional Fourier transform in (9) is exactly the row-column method of multi-dimensional Fourier transform computation. The modular implementation of (10) immediately suggests an efficient way of taking advantage of parallel and vector architectures of the target computer system. The stride parameter replaces the global permutation after each stage of DFT computation; The looping parameter replaces calling the same subroutine many times.

B.3 Cooley-Tukey FFT Algorithms

Suppose $N = LM$. The Cooley-Tukey algorithm (decimation-in-frequency) for one-dimensional Fourier transform is given by tensor product:

$$F(N) = P(N, M)(I_L \otimes F(M))T_M(N)(F(L) \otimes I_M), \tag{11}$$

where $P(N, M)$ is a $N \times N$ stride-M permutation matrix, $T_M(N)$ is a $N \times N$ block diagonal matrix of twiddle factors,

$$T_M(N) = \bigoplus_{l=0}^{L-1} D_M(N), \tag{12}$$

where

$$D_M(N) = diag.(1, w, \cdots, w^{M-1}). \tag{13}$$

The Cooley-Tukey FFT algorithm given in (11) can be used in an inductive argument to derive extension to many factors. Suppose

$$N = N_1 N_2 \cdots N_K. \tag{14}$$

Set $N(0) = 1$ and

$$N(k) = N_1 N_2 \cdots N_k, \quad 1 \leq k \leq K, \tag{15}$$

$$N'(k) = N/N(k), \quad 0 \leq k \leq K. \tag{16}$$

Define

$$F'_k = T_k(I_{N(k-1)} \otimes F(N_k) \otimes I_{N'(k)}), \tag{17}$$

where T_k is a diagonal matrix

$$T_k = I_{N(k-1)} \otimes T_{N'(k)}(N'(k-1)). \tag{18}$$

Then we have the Cooley-Tukey FFT algorithm for many factors:

$$F(N) = QF'_K \cdots F'_2 F'_1, \tag{19}$$

where Q is the generalized bit-reversal permutation matrix.

Each stage of Fourier transform F'_k, $1 \leq k \leq K$ with the twiddle factor multiplication can be written as:

$$I_{N(k-1)} \otimes (T_{N'(k)}(N'(k-1))(F(N_k) \otimes I_{N'(k)}). \tag{20}$$

(20) can be implemented as a module

$$I_{N(k-1)} \otimes [\bigoplus_{j=0}^{N'(k)-1} D^j_{N'(k)}(N'(k-1))(F(N_k) \otimes I_{N'(k)})]. \tag{21}$$

The parameters of this module are $N'(k-1)$, $N(k-1)$ and twiddle factors $D^j_{N'(k)}(N'(k-1))$. Although the form (21) does not look as neat as (10), the implementation is as easy. The twiddle factors are introduced into the module that varies with the stride parameter. Thus any large size Fourier transform computation is made by putting together a set of modules given in (10) and (21).

B.4 Multi-Dimensional FFT Algorithms

In this section, we will show that various multi-dimensional DFT algorithms can be unified into the format described in the previous sections: they can be decomposed into identifiable basic building blocks of small size modules.

Row-Column Method

Consider the 2-dimensional Fourier transform of (5). The row-column method of computing $\underline{y}$ is written as:

$$\underline{y} = (F(N_2) \otimes I_{N_1})(I_{N_2} \otimes F(N_1))\underline{x}. \tag{22}$$

Suppose $N_1 = L_1 M_1$ and $N_2 = L_2 M_2$. Using the Cooley-Tukey FFT algorithm of (11) into (22), we have

$$\begin{aligned} &((P(N_2, M_2)(I_{L_2} \otimes F(M_2))T_{M_2}(N_2) \\ &\times (F(L_2) \otimes I_{M_2})) \otimes I_{N_1})(I_{N_2} \otimes (P(N_1, M_1) \\ &\times (I_{L_1} \otimes F(M_1))T_{M_1}(N_1)(F(L_1) \otimes I_{M_1}))) \end{aligned} \tag{23}$$

The implementation of 2-dimensional DFT in (23) has the same structure as 1-dimensional FFT in (19). Two sets of DFT modules are computed; one with twiddle factor multiplications,

$$I_{Lm} \otimes [\bigoplus_{j=0}^{Ln-1} D^j_{L_i}(N_i)(F(L_i) \otimes I_{Ln})], \tag{24}$$

$i = 0, 1$, $0 \leq j < M_i$, and Ln and Lm are the parameters controlling the decimation and looping, and twiddle factor parameters come from $D^j_{L_i}(N_i)$;

The other module without twiddle factors;

$$I_{Lk} \otimes F(M_i) \otimes I_{Li}, \tag{25}$$

Lk and Li are the parameters of the module.

Vector-Radix Method

The vector-radix Cooley-Tukey FFT algorithm to compute (22) is given in the following factorization:

$$F(N_2) \otimes F(N_1) = PF_2'TF_1', \tag{26}$$

where

$$T = T_{M_2}(N_2) \otimes T_{M_1}(N_1), \tag{27}$$

$$P = P(N_2, M_2) \otimes P(N_1, M_1), \tag{28}$$

$$\begin{aligned} F_1' &= F(L_2) \otimes I_{M_2} \otimes F(L_1) \otimes I_{M_1} \\ &= (F(L_2) \otimes I_{M_2 L_1 M_1}) \\ &\quad (I_{L_2 M_2} \otimes F(L_1) \otimes I_{M_1}), \end{aligned} \tag{29}$$

$$\begin{aligned} F_2' &= I_{L_2} \otimes F(M_2) \otimes I_{L_1} \otimes F(M_1) \\ &= (I_{L_2} \otimes F(M_2) \otimes I_{L_1 M_1}) \\ &\quad (I_{L_2 M_2 L_1} \otimes F(M_1)), \end{aligned} \tag{30}$$

(26) can be computed by using the modules without twiddle factors and a separate stage of stride permutation of P and a twiddle factor multiplication stage T.

B.5 Implementation on Intel i860 Processor

In this section, we give an example of carrying out the computation of multi-dimensional DFT using our tensor product modules. Take the case 40×40 2-dimensional Fourier transform. Set $40 = 5 \times 8$. The tensor product form of the Cooley-Tukey FFT algorithm (row-column method) is

$$\begin{aligned} &F(40,40) = \\ &((P(40,5)(I_8 \otimes F(5))T_5(40)(F(8) \otimes I_5)) \otimes I_{40}) \\ &\times (I_{40} \otimes (P(40,5)(I_8 \otimes F(5))T_5(40)(F(8) \otimes I_5)) \end{aligned} \tag{31}$$

Variants can be derived from (31). One of them is

$$F(40,40) =$$

$$(P(40,5) \otimes I_{40})(I_8 \otimes F(5) \otimes I_{40})((T_5(40)(F(8) \otimes I_5)) \otimes I_{40})$$
$$\times (I_{40} \otimes P(40,5))(I_{320} \otimes F(5)))(I_{40} \otimes (T_5(40)(F(8) \otimes I_5))) \quad (32)$$

Both forms have their advantages. For the Intel i860 processor, algorithm (31) gives rise to faster implementation because it minimizes the cache thrash. The implementation of (31) is given as:

```
c transform on the columns
   do i=0,39
     call ftc8tw( x(0,i), 5, 1, 1, w, isign )
     call ftc5( x(0,i), y(0,i), 1, 8, 1, isign )
     call transpose(y, x)
   end do
c transform on the rows
   ⋮
```

The implementation of (32) is given as:

```
c transform on the columns
     call ftc8tw( x, 5, 1, 40, w, isign )
     call ftc5( x, y, 1, 8, 40, isign )
c transform on the rows
     call ftc8tw( y, 5*40, 40, 1, w, isign )
     call ftc5( y, x, 40, 8*40, 1, isign )
```

The module *ftc8tw* computes

$$I_{n_1} \otimes [\bigoplus_{j=0}^{n_2-1} D_5^j(40)(F(8) \otimes I_{n_2})], \quad (33)$$

isign denotes the forward or reverse transform, w denotes pre-calculated twiddle factors, and the module *ftc5* computes

$$I_{n_1} \otimes F(5) \otimes I_{n_2}. \quad (34)$$

The timing results of some of the one- and 2-dimensional Fourier transform are given in tables 1 and 2. They are compared to the Kuck and Associates, Inc. Math Library Package on the Intel iPSC/860. It is worth mentioning that Intel's 1-dimensional and 2-dimensional FFT routine are hand coded assembly program, while the AwareTime are the hybrid of Fortran calls and i860 hand coded assembly modules.

Table 1. Timing Results on i860 Processor(1-D)

FT Size N	AwareTime ms.	IntelTime[6] ms.
3	0.000363	
4	0.000449	0.0119
5	0.000881	
7	0.00164	
8	0.00139	0.0141
16	0.007	0.0191
20	0.0098	
32	0.0133	0.031
40	0.0211	
64	0.028	0.065
80	0.0528	
384	0.296	
512	0.350	0.560

ms. $= 10^{-3}$ second.

Table 2. Timing Results on i860 Processor(2-D)

FT Size $n \times n$	AwareTime	IntelTime
32×32		1.386 ms.
40×40	2.400 ms.	
64×64		6.65 ms.
80×80	12.9 ms.	
128×128		24.7 ms.
160×160	58.6 ms	
256×256		137 ms.
384×384	296 ms	
512×512		777 ms.

B.6 References

[1] C. Lu, M. An, Z. Qian and R. Tolimieri, "Small FFT Module Implementation on the Intel i860 Processor", the proceedings of ICSPAT'92, Cambridge, MA.

[2] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes: the Art of Scientific Computing*, Cambridge University Press, 1986.

[3] M. An, C. Lu, E. Prince and R. Tolimieri, "Fast Fourier Transforms for Space Groups Containing Rotation Axes of Order Three and Higher", *Acta Cryst.*, (1992), **A48**, 346-349.

[4] M. An, C. Lu, E. Prince and R. Tolimieri, "Fast Fourier Transforms for Real and Hermitian Symmetric Data ", *Acta Cryst.*, (1992), **A48**, 415-418.

[5] G. Bricogne and R. Tolimieri, "Symmetrized FFT Algorithms", *IMA, Math. and Its Appl., 23, Springer-Verlag, New York/Berlin, 1990.*

[6] The Kuck and Associates, Inc. Math Library on the Intel iPSC/860.

[7] M. C. Pease, "An Adaption of the Fast Fourier Transform for Parallel Processing", *Journal of the ACM*, April, 1968.

[8] R. Tolimieri, M. An and C. Lu, *Algorithms for Discrete Fourier Transform and Convolutions*, Springer-Verlag. 1989.

[9] R. Tolimieri, M. An and C. Lu, *The Mathematics of Multi-dimensional Fourier Transform Algorithms*, Springer-Verlag, in printing, 1993.

[10] Lu C., Cooley, J.W. and Tolimieri, R. (1992),"FFT Algorithms for Prime Transform Sizes and Their Implementations on VAX, IBM 3090VF and RS/6000", *IEEE Trans. on Signal Processing*, Feb, 1993.

[11] Margulis, N. (1990), *i860 Microprocessor Architecture*, McGraw-Hill.

[12] *IBM Journal of Research and Development: Special Issue on IBM RISC System/6000 Processor*, June, 1990.

[13] *iPSC/860 Supercomputer Advanced Information Fact Sheet.* Intel 1990.

C A New Approach for Computing Multi-dimensional DFTs on Parallel Machines and its Implementation on the iPSC/860 Hypercube

Abstract

In this paper we propose a new approach for computing multi-dimensional DFTs that reduces interprocessor communications and is therefore suitable for efficient implementation on a variety of multiprocessor machines. Group theoretic concepts are used to formulate a computational strategy that hybrids the Reduced Transform Algorithm (RTA) with the Good-Thomas factorization. The RTA algorithm is employed not as a data processing but rather as a book-keeping tool in order to decompose the problem into many smaller size sub-problems that can be solved independently. Implementation issues on an Intel iPSC/860 hypercube are discussed and timing results are provided for many different cases. The non-optimized realizations of the new approach are shown to outperform the highly optimized realizations of the traditional row-column method in a variety of test cases.

C.1 Introduction–Motivation

Parallel computing presents a new environment for algorithm design and implementation, along with new challenges to the computational scientist. The performance of any given program depends on an increased number of parameters compared to the serial case, widening this way the difference between theoretical models and practical experience.

In this paper, we present a strategy for computing a multidimensional DFT that hybrids a relatively new algorithm (Reduced Transform Algorithm) with already implemented single processor kernel routines. We will use the reduced transform algorithm to address the reduction and optimization of interprocessor communications. Our work has been mainly motivated from the distributed memory parallel computing paradigm, which is arguably the most difficult to harness due to its exposed interprocessor communication to the programmer. Most parallel computers require sophisticated algorithms and programming techniques for their optimum utilization. In this discussion, we will make use of algebraic facts in presenting the algorithms. The parameters in algebraic formulas give us the important implementation parameters. Thus the flexibility to address the variables in implementations is equated with flexibility in manipulating algebraic formalism. Initial investment in familiarity with some amount of algebra may be necessary, but the payoff is immediate. Most of the relevant algebra, not in its most rigorous form but its usage, can be found

in [1].

In its most general form, the *Reduced transform algorithm* (RTA) is a full utilization of the duality between periodic and decimated data in the Fourier transform. This duality was used partially in some algorithms and implementations for restricted cases [2, 3, 4, 5]. A description of a generalization in a unified setting is found in [6, 7] along with the work of M. Rofheart [8]. In this paper, we will consider the application of RTA to the case $\mathbf{Z}/P \times \mathbf{Z}/P$, for a prime number P. Tensor product formulation of DFT computation on $\mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$ is interleaved with the periodization step in RTA for $\mathbf{Z}/P \times \mathbf{Z}/P$ to produce $P+1$ independent data of size NP.

We will use the RTA to address the imbalance between computation and communication rates in current distributed memory parallel machines by reducing communication between processors to collective patterns only (broadcast and combine) instead of the all-to-all communication patterns required in the global matrix transpose needed by the row-column (RC) implementations of multidimensional DFT's. Also, since fast algorithms for prime size 1D-DFT's exist [1] and the case $\mathbf{Z}/P \times \mathbf{Z}/P$ of the RTA is very efficient because its computation requires only $P+1$ 1D transforms (versus $2P$ for the row column method), our approach addresses the issue of storage reduction by providing additional transform size options. For example the ability to perform a 181×181 point 2D DFT means potential storage savings up to 50% over the 256×256 case, along with the savings in computational time. The storage savings can be used for the optimization of the broadcasting step needed for the RTA, in environments with long communications latency.

Via the Chinese remainder theorem, we will extend our method to compute the 3-dimensional DFT on $\mathbf{Z}/N \times \mathbf{Z}/MP \times \mathbf{Z}/KP$, where N is an arbitrary integer, M and K are integers not divisible by P, for a prime P. We transform the data set to an equivalent 5D data set on $\mathbf{Z}/N \times \mathbf{Z}/M \times \mathbf{Z}/K \times \mathbf{Z}/P \times \mathbf{Z}/P$, and then employs the RTA on the last two indices to break the problem into smaller independent sub-problems that can be computed in parallel. Each sub-problem is associated with the computation of the value of the Fourier Transform along one *line* in the set $\mathbf{Z}/P \times \mathbf{Z}/P$ passing through the origin. These lines intersect only at the origin and cover the index space. When translated from the 5D data set back to the original 3D data, each line corresponds to a set of parallel lines covering the index space.

Three stages are needed to compute the values of the DFT along the lines: (1) Periodization stage, which consists of additions of data along lines perpendicular to a given line, (2) 3D Cooley-Tuckey FFT and (3) P-point DFT. In a multiprocessor environment, each processor computes these three steps independently of the others thus allowing for maximum parallelism and efficiency. Moreover, the final data distribution among the processors is such as to permit further processing in a parallel fashion since every processor holds only results belonging to the same geometrical

subset.

The proposed hybrid method (HRTA) can be used in applications such as the computation of motion from a sequence of images (multi-frame detection, MFD), a very important task in computer vision, HDTV and video telephony. Several methods for MFD have been proposed in the literature that are usually divided into two categories: *Time Domain* methods, that estimate the motion by processing the sequence of images directly, and the recently proposed *Frequency Domain* methods [9], [10] that processes the frequency contents of the images to estimate the velocity and trajectory of the moving components. The latter methods offer more robust detection and huge computational savings since the frequency domain representation of the 3D data (sequence of 2D images) is more compact than the equivalent time domain representation. With all the processors holding data belonging to different lines in the frequency domain, each processor can independently test for the presence of motion along its assigned direction.

This paper is organized as follows: In section 2 we describe the RTA with an application on $\mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$ and its parallel processing strategy. In section 3 we discuss the extension via the Chinese remainder theorem and introduce hybrid algorithm (HRTA) that we use on $\mathbf{Z}/N \times \mathbf{Z}/MP \times \mathbf{Z}/KP$, and its parallel variant. In section 4 we discuss issues related to the implementation of the hybrid algorithm on the Intel iPSC/860 parallel machine. In section 5 we present detailed timing results and a thorough comparison of our approach with the traditional row-column method for a variety of 2D and 3D DFT cases. We close the paper in section 6 by summarizing our findings and propose directions for further investigation.

C.2 The Reduced Transform Algorithm (RTA) on $\mathbf{Z}/P \times \mathbf{Z}/P$

Before we proceed we need the following definitions:
Let G be an abelian group of the form

$$G = \mathbf{Z}/N_1 \times \mathbf{Z}/N_2 \times \cdots \times \mathbf{Z}/N_R.$$

For $\mathbf{g}, \mathbf{h} \in G$, define the bilinear map from G to C^x the complex numbers of magnitude 1 by

$$\chi(\mathbf{g}, \mathbf{h}) = e^{-2\frac{\pi i}{N_1} g_1 h_1} e^{-2\frac{\pi i}{N_2} g_2 h_2} \cdots e^{-2\frac{\pi i}{N_R} g_R h_R}, \tag{1}$$

where $\mathbf{g} = (g_1, g_2, \cdots, g_R)$, $\mathbf{h} = (h_1, h_2, \cdots, h_R)$. Since $g_r h_r$, $1 \leq r \leq R$, is uniquely defined in $\mathbf{Z}/N_r$, (1) is well defined. For a subgroup S of G, the *dual* of S, denoted by $S^\perp$ is the following subgroup of G.

$$S^\perp = \{\mathbf{g} \in G : \chi(\mathbf{g}, \mathbf{s}) = 1, \text{for all } s \in S\}.$$

In addition to duality, we will use the following definition. Let S be a subgroup of an abelian group G. A subgroup S^c of G is called a *complementary* subgroup of S if every element $g \in G$ can be written as

$$\mathbf{g} = \mathbf{s} + \mathbf{c}, \quad \mathbf{s} \in S, \mathbf{c} \in S^c.$$

In general, the complementary subgroup is not unique. Moreover, not every subgroup has a complementary subgroup.

Let $G = \mathbf{Z}/P \times \mathbf{Z}/P$, where P is a prime number. Non-trivial subgroups of G are of order P, and hence cyclic. In addition, every subgroup of G has a complementary subgroup. The Reduced Transform Algorithm (RTA) on G proceeds as follows:

1. DETERMINE OUTPUT DECIMATING SUBGROUPS TO COVER G:
For $0 \leq l < P$ set:

$$\mathbf{P}_l = \{a(1,l) : 0 \leq a \leq P-1\},$$

and for $l = P$ set:

$$\mathbf{P}_P = \{(0,a) : 0 \leq a \leq P-1\}.$$

We have

$$\bigcup_{l=0}^{P} \mathbf{P}_l = G.$$

2. DETERMINE THE INPUT PERIODIZING SUBGROUPS, FOR $0 \leq l \leq P$.
Denote by $\mathbf{Q}_l$, $0 \leq l < P$, the following subgroups of G.

$$\mathbf{Q}_l = \{b(-l,1) : 0 \leq b < P\}.$$

Also for $l = P$

$$\mathbf{Q}_P = \{b(1,0) : 0 \leq b < P\}.$$

$\mathbf{Q}_l$, $0 \leq l \leq P$, is a subgroup of order P and $\mathbf{Q}_l = \mathbf{P}_l^{\perp}$.

In Figure 1 we show the output decimating subgroups $\mathbf{P}_l$ for $G = \mathbf{Z}/P \times \mathbf{Z}/P$, $P = 3$ (area inside the box). If we extend the index space, each $\mathbf{P}_l$ corresponds to a "line". Due to the modulo P operations certain points of a line outside the box (marked with a circled $+$) will be mapped inside (to the corresponding circled node with $\star$ in the same row/column). Also note that due to the periodicity, the two lines labeled P_2 are actually the same. All lines intersect at the origin. In the same figure we show the input periodizing subgroups $\mathbf{Q}_l$. The collection $\mathbf{Q}_l$ of input lines cover the whole index space, as the collection $\mathbf{P}_l$ of output lines do, and are dual to them.

3. COMPUTE THE PERIODIZATIONS.

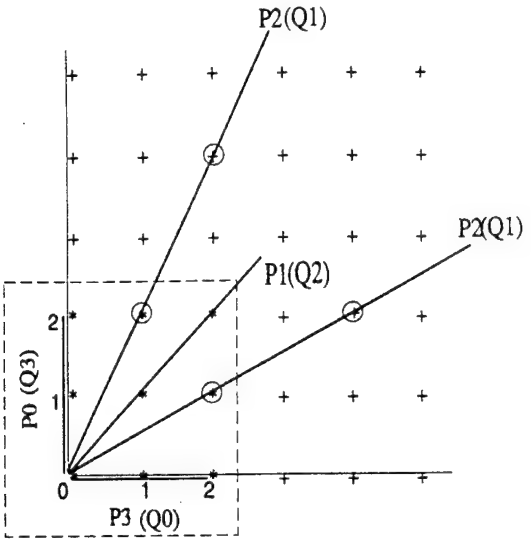


Figure 1: The output decimating subgroups (lines) $\mathbf{P}_l$ and the input periodizing subgroups (lines) $\mathbf{Q}_l$ for the case $P = 3$.

A periodization is completely determined by its values on a complementary group. Fix a complementary subgroup for $\mathbf{Q}_l$, $0 \le l \le P$, and denote it by $\mathbf{Q}_l^c$.

$$g_l(\mathbf{c}) = \sum_{\mathbf{b} \in \mathbf{Q}_l} f(\mathbf{c} + \mathbf{b}), \quad 0 \le l \le P, \mathbf{c} \in \mathbf{Q}_l^c. \tag{2}$$

Although there are many choices for complementary subgroups, we will fixed them to be:

$$\mathbf{Q}_l^c = \{(c, 0) : 0 \le c \le P - 1\}, \quad 0 \le l < P$$

$$\mathbf{Q}_P^c = \{(0, c) : 0 \le c \le P - 1\}, \quad l = P$$

4. COMPUTE THE DFT. For $\mathbf{a} \in \mathbf{P}_l$,

$$\widehat{f}_l(\mathbf{a}) = \sum_{\mathbf{c} \in \mathbf{Q}_l^c} \sum_{\mathbf{b} \in \mathbf{Q}_l} f(\mathbf{c} + \mathbf{b}) \chi(\mathbf{c} + \mathbf{b}, \mathbf{a}).$$

Since $\chi(\mathbf{b}, \mathbf{a}) = 1$, using (2) we get

$$\widehat{f}_l(\mathbf{a}) = \sum_{\mathbf{c} \in \mathbf{Q}_l^c} g_l(\mathbf{c}) \chi(\mathbf{c}, \mathbf{a}). \tag{3}$$

For $0 \le l \le P - 1$, we will use the following identification to index the computations,

$$\mathbf{a} \longleftrightarrow (a, al), \quad \mathbf{b} \longleftrightarrow (-bl, b), \quad \mathbf{c} \longleftrightarrow (c, 0),$$

$$0 \le a, b, c < P, \quad \mathbf{a} \in \mathbf{P}_l, \mathbf{b} \in \mathbf{Q}_l, \mathbf{c} \in \mathbf{Q}_l^c.$$

For $l = P$, the identification is:

$$\mathbf{a} \longleftrightarrow (0,a), \quad \mathbf{b} \longleftrightarrow (b,0), \quad \mathbf{c} \longleftrightarrow (0,c),$$

$$0 \le a, b, c < P \qquad \mathbf{a} \in \mathbf{P}_l, \mathbf{b} \in \mathbf{Q}_l, \mathbf{c} \in \mathbf{Q}_l^c.$$

Therefore we can rewrite (2) and (3) as follows.

$$g_l(c) = \sum_{b=0}^{P-1} f(c - bl, b), \qquad 0 \le l < P, \qquad 0 \le c < P$$

$$g_P(c) = \sum_{b=0}^{P-1} f(b, c), \qquad 0 \le c < P$$

$$\widehat{f}(a, al) = \sum_{c=0}^{P-1} g_l(c) e^{-\frac{2\pi i}{P} ac} \qquad 0 \le l < P, \qquad 0 \le c < P$$

$$\widehat{f}(0, a) = \sum_{c=0}^{P-1} g_P(c) e^{-\frac{2\pi i}{P} ac}, \qquad 0 \le c < P$$

C.2.1 Application: The case $A = \mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$

Let $A = \mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$, for a natural number N and a prime number P. For $f \in L(A)$ and $(u, v, w) \in A$, the Fourier transform, $\widehat{f}$, is defined by

$$\widehat{f}(u,v,w) = \sum_{z=0}^{P-1} \sum_{y=0}^{P-1} \sum_{x=0}^{N-1} f(x,y,z) e^{-2\pi i \frac{ux}{N}} e^{-2\pi i \frac{vy}{P}} e^{-2\pi i \frac{wz}{P}}.$$

For $\mathbf{a} \in \mathbf{P}_l$, $0 \le l \le P$,

$$\widehat{f}_l(u, \mathbf{a}) = \sum_{\mathbf{c} \in \mathbf{Q}_l^c} \sum_{\mathbf{b} \in \mathbf{Q}_l} \sum_{x=0}^{N-1} f(x, \mathbf{b} + \mathbf{c}) e^{-2\pi i \frac{ux}{N}} \omega^{<\mathbf{a},\mathbf{c}>},$$

where $\omega = e^{-\frac{2\pi i}{P}}$, and $<\mathbf{a}, \mathbf{c}>$ corresponds to the usual inner product. Changing the order of summation,

$$\widehat{f}_l(u, \mathbf{a}) = \sum_{\mathbf{c} \in \mathbf{Q}_l^c} \left[\sum_{x=0}^{N-1} \left(\sum_{\mathbf{b} \in \mathbf{Q}_l} f(x, \mathbf{c} + \mathbf{b}) \right) e^{-2\pi i \frac{ux}{N}} \right] \omega^{<\mathbf{a},\mathbf{c}>}.$$

or equivalently,

$$\widehat{f}_l(u, \mathbf{a}) = \sum_{\mathbf{c} \in \mathbf{Q}_l^c} \sum_{x=0}^{N-1} g_l(x, \mathbf{c}) e^{-2\pi i \frac{ux}{N}} \omega^{<\mathbf{a},\mathbf{c}>}.$$

This computation can also be rewritten, using our identification scheme as follows:
For $0 \leq l \leq P-1$,

$$\widehat{f}(u,a,al) = \sum_{c=0}^{P-1}\sum_{x=0}^{N-1} g_l(x,\mathbf{c})e^{-2\pi i\frac{ux}{N}}\omega^{ac} = \sum_{c=0}^{P-1}\sum_{x=0}^{N-1} g_l(x,0,c)e^{-2\pi i\frac{ux}{N}}\omega^{ac},$$

and for $l = P$,

$$\widehat{f}(u,0,a) = \sum_{c=0}^{P-1}\sum_{x=0}^{N-1} g_l(x,\mathbf{c})e^{-2\pi i\frac{ux}{N}}\omega^{ac} = \sum_{c=0}^{P-1}\sum_{x=0}^{N-1} g_l(x,c,0)e^{-2\pi i\frac{ux}{N}}\omega^{ac}.$$

In Figure 2 we depict the three-dimensional index set $A = \mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$ in which planes defined by the last two indices are partitioned into lines. In essence, the algorithm can be thought of as N times the RTA on data sets: $\mathbf{Z}/P \times \mathbf{Z}/P$.

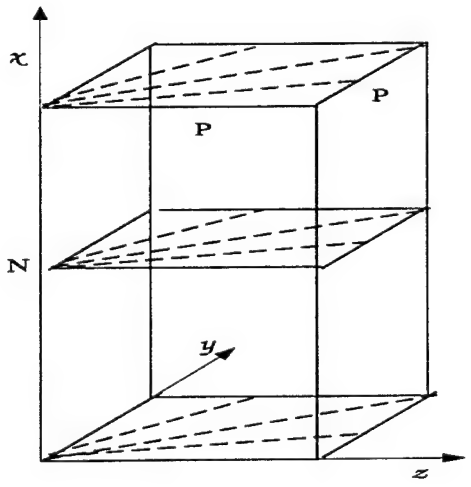


Figure 2: The 3D index set, partitioned into lines along the last two dimensions, and into parallel planes along the first dimension.

C.2.2 The parallel processing strategy

In the previous subsection we have shown how the DFT on a 3D index set can be partitioned to independent computations. For each one of the $P+1$ lines of a plane, P periodizations needs to be computed for a total of $N \cdot (P+1) \cdot P$ periodizations. Depending on the number of processors (PEs) and the available memory per processor, different parallel implementations can be derived. If $P+1$ PEs are available, each one can be assigned to compute the DFT on one of the lines, and there is no need for interprocessor communications. This scheme however requires that each PE has access to the whole data set and is able to store at least the periodized data along a whole line for all values of x $(N \cdot P$ elements).

Alternatively if $P \cdot (P+1)$ processors are available, each one may be assigned to compute the values of the DFT for one of the P points that belong to a particular line. For such an implementation, the minimum memory requirements for a node is reduced to N, and more parallelism is exploited at the expense of some inter-processor communications. The parallel processing strategy is summarized below.

step 1: *Compute in parallel the* $N(P^2+P)$ *Periodizations*

$$g_l(x,\mathbf{c}) = \sum_{\mathbf{b}\in\mathbf{Q}_l} f_l(x,\mathbf{c}+\mathbf{b}), \tag{4}$$

$$= \begin{cases} \sum_{b=0}^{P-1} f(x, c-bl, b), & 0 \le l \le P-1, \\ \sum_{b=0}^{P-1} f(x,b,c), & l = P. \end{cases}$$

• If $P+1$ PEs are used:
PE_l, $l = 0,\ldots,P$, computes the $N \cdot P$ periodizations $\{g_l(x,\mathbf{c}),\ 0 \le x < N,\ 0 \le c < P\}$. No interprocessor communications are required.

• If P^2+P PEs are used:
$PE_{l,c}$, $l = 0,\ldots,P$, $c = 0,\ldots,P-1$, computes the N periodizations $\{g_l(x,\mathbf{c}),\ 0 \le x < N\}$. Since the summation in (4) extends over $0 \le b \le P-1$ $PE_{l,c}$ needs to receive data residing in each $PE_{l,\gamma}$, $\gamma \ne c$.

step 2: *Compute the 1D, N-point DFTs.*

$$\widehat{g}_l(u,\mathbf{c}) = \sum_{x=0}^{N-1} g_l(x,\mathbf{c}) e^{-2\pi i \frac{ux}{N}}.$$

• If $P+1$ then PE_l computes the P, 1D N-point DFTs $\{\widehat{g}_l(u,\mathbf{c}),\ 0 \le c < P\}$.
• If P^2+P then $PE_{l,c}$ computes an 1D N-point DFT, namely $\widehat{g}_l(u,\mathbf{c})$.
No interprocessor communications are required in either cases.

step 3: *Compute the P-point 1D DFTs.*

$$\widehat{f}(u,\mathbf{a}) = \sum_{\mathbf{c}\in\mathbf{Q}_l^c} \widehat{g}_l(x,\mathbf{c}) w^{<\mathbf{a}.\mathbf{c}>}. \tag{5}$$

• If $P+1$ then PE_l computes the N, 1D P-point DFTs $\{\widehat{f}(u,\mathbf{a}),\ 0 \le u < N\}$. No interprocessor communications are required.
• If P^2+P then $PE_{l,c}$ computes an 1D P-point DFT, namely $\widehat{f}(u,\mathbf{a})$. Since the summation in (5) extends over $0 \le c \le P-1$ $PE_{l,c}$ needs to receive the partial result $\widehat{g}_l(x,\gamma)$ from each $PE_{l,\gamma}$, $\gamma \ne c$.

C.3 Extension via the Chinese Remainder Theorem

The Chinese Remainder theorem is a major tool in algorithm design. It is the basis of the *prime factor algorithm* of Good and Thomas [11, 12]. It can be stated in several ways, but we will use the theorem as a statement about rings, especially the idempotents, for uniformity and predictability in implementation.

C.4 Extension via the Chinese Remainder Theorem

The Chinese Remainder theorem is a major tool in algorithm design. It is the basis of the *prime factor algorithm* of Good and Thomas [11, 12]. It can be stated in several ways, but we will use the theorem as a statement about rings, especially the idempotents, for uniformity and predictability in implementation.

Theorem 1 Chinese Remainder Theorem [13].

Let $N = N_1 N_2$, where the integers N_1 and N_2 are relatively prime. Then

$$\mathbf{Z}/N \simeq \mathbf{Z}/N_1 \times \mathbf{Z}/N_2.$$

Rather than proving the theorem, we state an explicit isomorphism and its inverse. The mapping $\psi : \mathbf{Z}/N \to \mathbf{Z}/N_1 \times \mathbf{Z}/N_2$ defined by:

$$\psi(n) = (n \bmod N_1,\ n \bmod N_2)$$

is an isomorphism. The inverse is defined in terms of the *idempotents*. Let e_1, e_2 be the elements of $\mathbf{Z}/N_1 N_2$ with

$$\psi(e_1) = (1,0), \quad \psi(e_2) = (0,1).$$

Then the mapping defined below is ψ^{-1}.

$$\mathbf{Z}/N_1 \times \mathbf{Z}/N_2 \to \mathbf{Z}/N : (n_1, n_2) \mapsto (e_1 n_1 + e_2 n_2) \bmod N.$$

C.4.1 Good-Thomas Prime Factor Algorithm for $\mathbf{Z}/MP$

Henceforth we will restrict to the case where N_2 is a prime number. Set $N_2 = P$ and $N_1 = M$. The system of idempotents in this case will be given according to the residue of M by P.

Theorem 2 *Let $M \equiv c \bmod P$. Then $e_2 = c^{-1}M$, where c^{-1} is the inverse of $c \in U(\mathbf{Z}/P)$, the multiplicative group of units of $\mathbf{Z}/P$.*

Proof: $M = c + mP$ for some $m \in \mathbf{Z}$.

$$c^{-1}M = c^{-1}(c + mP) \equiv 1 \; mod \; P, \quad c^{-1}M \equiv 0 \; mod \; M.$$

Thus $\psi(c^{-1}M) = (0,1)$. Since $e_1 + e_2 \equiv 1 \; mod \; MP$, we have that $e_1 = MP + 1 - e_2$.

Example: 3 and 5 are relatively prime to each other. We will find the idempotents for the isomorphism $\mathbf{Z}/15 \simeq \mathbf{Z}/3 \times \mathbf{Z}/5$. $3 \equiv 3 \; mod \; 5$. $3^{-1} \equiv 2 \in \mathbf{Z}/5$. Thus,

$$e_2 = 2 \cdot 3 = 6 \in \mathbf{Z}/15, \quad e_1 = 15 + 1 - e_2 = 10 \in \mathbf{Z}/15.$$

We also have the isomorphism $\mathbf{Z}/15 \simeq \mathbf{Z}/5 \times \mathbf{Z}/3$. $5 \equiv 2 \; mod \; 3$. $2^{-1} \equiv 2 \in \mathbf{Z}/3$. Thus,

$$e_2 = 2 \cdot 5 = 10 \in \mathbf{Z}/15, \quad e_1 = 15 + 1 - e_2 = 6 \in \mathbf{Z}/15.$$

Indexing $\mathbf{Z}/MP$ by the CRT, DFT on $\mathbf{Z}/MP$ is computed by $F(P) \otimes F(M)$, where $F(L)$ denotes the L-point DFT matrix and $\otimes$ denotes the tensor product of matrices. Many formulations of the Prime Factor Algorithm (PFA) exist [14, 15, 16, 17], but the explicit use of idempotents to arrive at the tensor product decomposition can can be found in [1, 18]. We will formulate the PFA for two factors directly here since the derivation is easy and understanding the role of idempotents has a direct impact on parallel implementation.

To derive the tensor product decomposition in the language of matrices, we will begin by describing two distinct orderings of the group $\mathbf{Z}/MP$. Let $\{e_1, e_2\}$ be the idempotents for the isomorphism $\mathbf{Z}/MP \simeq \mathbf{Z}/M \times \mathbf{Z}/P$. The following presentations for the elements of $\mathbf{Z}/MP$ are unique.

$$x \in \mathbf{Z}/MP, \quad x = me_1 + ae_2, \;\; 0 \leq m < M, \; 0 \leq a < P. \tag{6}$$

$$y \in \mathbf{Z}/MP, \quad y = \mu P + \alpha M, \;\; 0 \leq \mu < M, \; 0 \leq \alpha < P. \tag{7}$$

- Order $\mathbf{Z}/MP$ antilexicographically by the pair (m, a) obtained by the presentation of the elements of $\mathbf{Z}/MP$ given in (6). We will use this to order the input data.
- Order $\mathbf{Z}/MP$ antilexicographically by the pair (μ, α) obtained by the presentation of the elements of $\mathbf{Z}/MP$ given in (7). We will use this to order the output of the Fourier transform computation.

$$\widehat{f}(\mu P + \alpha M) = \sum_{a=0}^{P-1} \sum_{m=0}^{M-1} f(me_1 + ae_2) e^{-\frac{2\pi i}{MP}(\mu m P e_1 + \alpha a M e_2)}.$$

Recall that $e_1 \equiv 1 \; mod \; M$ and $e_2 \equiv 1 \; mod \; P$. Since

$$e^{-\frac{2\pi i}{MP}(\mu m P e_1 + \alpha a M e_2)} = e^{-\frac{2\pi i}{MP}\mu m P e_1} e^{-\frac{2\pi i}{MP}\alpha a M e_2} = e^{-\frac{2\pi i}{M}\mu m e_1} e^{-\frac{2\pi i}{P}\alpha a e_2} = e^{-\frac{2\pi i}{M}\mu m} e^{-\frac{2\pi i}{P}\alpha a}$$

we have that

$$\widehat{f}(\mu P + \alpha M) = \sum_{a=0}^{P-1} \sum_{m=0}^{M-1} f(me_1 + ae_2) e^{-\frac{2\pi i}{M}\mu m} e^{-\frac{2\pi i}{P}\alpha a}. \tag{8}$$

For a function f defined on $\mathbf{Z}/MP$, denote by $\underline{f}$ the vector of values $f(x)$ ordered by (6). Denote by $\underline{\widehat{f}}$ the vector of the Fourier transform of f ordered by (7). We can express (8) in terms of matrices as follows.

$$\underline{\widehat{f}} = [F(P) \otimes F(M)]\,\underline{f}.$$

C.4.2 The Hybrid Good-Thomas and RTA Algorithm on $A = \mathbf{Z}/N \times \mathbf{Z}/MP \times \mathbf{Z}/KP$

Let $A = \mathbf{Z}/N \times \mathbf{Z}/MP \times \mathbf{Z}/KP$, for natural numbers N, M, K and a prime number P such that $GCD(M,P) = GCD(K,P) = 1$. By applying the CRT twice, we have the isomorphism $A \simeq \mathbf{Z}/N \times \mathbf{Z}/M \times \mathbf{Z}/P \times \mathbf{Z}/K \times \mathbf{Z}/P$.

For $f \in L(A)$ and $(u,v,w) \in A$, the Fourier transform, $\widehat{f}$, is defined by

$$\widehat{f}(u,v,w) = \sum_{z=0}^{PK-1} \sum_{y=0}^{PM-1} \sum_{x=0}^{N-1} f(x,y,z) e^{-2\pi i \frac{ux}{N}} e^{-2\pi i \frac{vy}{MP}} e^{-2\pi i \frac{wz}{KP}}.$$

Set

$$g(n,m,k,a,b) = f(n, e_1 m + e_2 a, f_1 k + f_2 b), \tag{9}$$

where $\{e_1, e_2\}$ is the system of idempotents for the isomorphism $\mathbf{Z}/MP \simeq \mathbf{Z}/M \times \mathbf{Z}/P$ and $\{f_1, f_2\}$ is the system of idempotents for the isomorphism $\mathbf{Z}/KP \simeq \mathbf{Z}/K \times \mathbf{Z}/P$. We can compute $\widehat{f}$ by computing $\widehat{g}$ since

$$\widehat{f}(\nu, \mu P + \alpha M, \kappa P + \beta M) = \widehat{g}(\nu, \mu, \kappa, \alpha, \beta).$$

In the previous section, we described an algorithm for the case of an index set $A = \mathbf{Z}/N \times \mathbf{Z}/P \times \mathbf{Z}/P$. The same ideas can be applied to the index set $A \simeq \mathbf{Z}/N \times \mathbf{Z}/M \times \mathbf{Z}/P \times \mathbf{Z}/K \times \mathbf{Z}/P$. If N, M and K are powers of 2, the RTA algorithm can be used to decompose the data set into independent computations that can be performed on each of the $P+1$ (or P^2+P) processors. The algorithm remains essentially the same, with the 1D N-point DFT kernel now replaced by the 3D $N \times M \times K$ DFT kernel. The additional data re-indexing defined by equation (9) can be incorporated into the computation of the periodizations with respect to the sets $\mathbf{Z}/P \times \mathbf{Z}/P$ during the first step. In is interesting to note that with the application of the CRT, the resulting *hybrid* algorithm now computes the DFT on sets of lines that are *parallel* to the lines of Figure 1 as shown in Figure 3 for the case $P = 3$, $M = K = 2$.

C.4.3 The parallel hybrid algorithm using $P+1$ processors

The parallel algorithm for the computation of the 3D Fourier Transform of a complex function defined on the index set $A = \mathbf{Z}/N \times \mathbf{Z}/MP \times \mathbf{Z}/KP$ is given bellow:

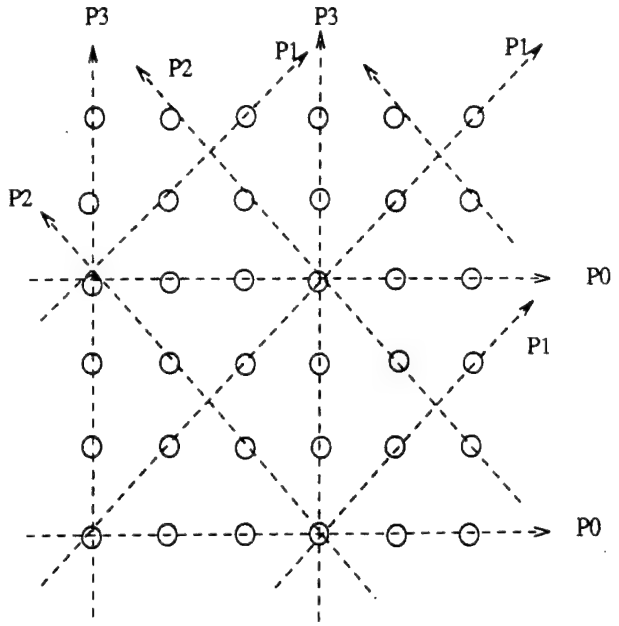


Figure 3: The output decimating lines for the case $P = 3$, $M = K = 2$.

Processor l $(l = 0, \ldots, P)$

- **step 1:** *Combined computation of Good-Thomas permutation and Periodizations*

  for $c = 0 \ldots P-1$, $b = 0 \ldots P-1$,

  for $n = 0 \ldots N-1$, $m = 0 \ldots M-1$, $k = 0 \ldots K-1$,

  if $(l < P)$ then

  $g_l(n,m,k,c) := g_l(n,m,k,c) + f(n, (e_1 m + e_2(c - bl)_P)_{MP}, (f_1 k + f_2 b)_{KP})$

  else

  $g_l(n,m,k,c) := g_l(n,m,k,c) + f(n, (e_1 m + e_2 b)_{MP}, (f_1 k + f_2 c)_{KP})$

  where we denote by $(\cdot)_E$ the *modulo* E operation. Note that at this step every processor needs to access the whole data set stored in the array $f(N, MP, KP)$ and at the end produces an $N \times M \times K \times P$ array containing the periodized data with respect to the line l.

- **step 2:** *Computation of P 3D FFTs of size* $N \times M \times K$

  for $c = 0 \ldots P-1$

  $\hat{g}_l(n,m,k,c) = \sum_{\hat{n}=0}^{N-1} \sum_{\hat{m}=0}^{M-1} \sum_{\hat{k}=0}^{K-1} g_l(\hat{n},\hat{m},\hat{k},c)) e^{\frac{-2\pi i n\hat{n}}{N}} e^{\frac{-2\pi i m\hat{m}}{M}} e^{\frac{-2\pi i k\hat{k}}{K}}$

- **step 3:** *Computation of* $(N \cdot M \cdot K)$, *1D P-point DFTs*

  for $n = 0 \ldots N-1$, $m = 0 \ldots M-1$, $k = 0 \ldots K-1$,

  $\hat{f}(n,m,k) = \sum_{\hat{c}=0}^{P-1} \hat{g}_l(n,m,k,\hat{c}) e^{\frac{-2\pi i c\hat{c}}{P}}$

Note that the same ideas can be also employed to compute the 2D DFT for a function defined on the index set: $A = \mathbf{Z}/MP \times \mathbf{Z}/KP$. The parallel hybrid algorithm for the case of $P+1$ nodes is given bellow:

Processor l $(l = 0, \ldots, P)$

- **step 1:** *Combined computation of Good-Thomas permutation and periodizations*

  for $c = 0 \ldots P-1, \quad b = 0 \ldots P-1,$

  for $m = 0 \ldots M-1, \quad k = 0 \ldots K-1,$

  if $(l < P)$ then

  $$g_l(m,k,c) := g_l(m,k,c) + f((e_1 m + e_2(c - bl)_P)_{MP}, (f_1 k + f_2 b)_{KP})$$

  else

  $$g_l(m,k,c) := g_l(m,k,c) + f((e_1 m + e_2 b)_{MP}, (f_1 k + f_2 c)_{KP})$$

  The array $g_l(M,K,P)$ contains now the periodized data with respect to the line l.

- **step 2:** *Computation of P 2D FFTs of size $M \times K$*

  for $c = 0 \ldots P-1$ compute

  $$\hat{g}_l(m,k,c) = \sum_{\hat{m}=0}^{M-1} \sum_{\hat{k}=0}^{K-1} g_l(\hat{m},\hat{k},c)) e^{\frac{-2\pi i m \hat{m}}{M}} e^{\frac{-2\pi i k \hat{k}}{K}}$$

- **step 3:** *Computation of $(M \cdot K)$ P-point DFTs*

  for $m = 0 \ldots M-1, \quad k = 0 \ldots K-1,$ compute

  $$\hat{f}(m,k) = \sum_{\hat{c}=0}^{P-1} \hat{g}_l(m,k,\hat{c}) e^{\frac{-2\pi i c \hat{c}}{P}}$$

C.5 Implementation issues

C.5.1 The Intel iPSC/860 Hypercube

The Intel iPSC/860 parallel processing system is a distributed memory, Multiple Instruction Multiple Data (MIMD) hypercube, containing up to $128 = 2^7$ compute nodes (processing elements, PEs) based on the Intel i860 high performance 64-bit RISC microprocessor. The i860 has a peak performance of 80 MFlops and is equipped with 8K data and 4K instruction cache memory. Each node has 8 to 64 Mbytes of external local memory, a network interface and a message router. The router can handle up to 8 bidirectional communication channels, seven of which may be connected to neighboring nodes and one is dedicated to external I/O and is directly connected to the host processor.

The PEs are connected to each other via relatively slow full duplex asynchronous communication channels that can carry messages of variable length. The channel bandwidth is about 2.8Mbytes/sec. The *wormhole* routing technique, which minimizes the delay between receiving a message in a node and retransmitting it to its final destination, is used. The message passing can

be either *synchronous* or *asynchronous*. The synchronous message passing blocks the execution of the node programs until the communication has been completed, whereas asynchronous message passing returns immediately and is useful if the node processors can perform other computations while waiting for the communication to complete. The system is equipped with a Concurrent File System (CFS) [19] that distributes files across all available disks in blocks, such that different compute nodes can access different parts of a file without creating a bottleneck at a particular I/O node.

C.5.2 Initial data loading and distribution

The hybrid Reduced Transform Algorithm (HRTA) that we propose requires that the whole data set is accessible from all the nodes, so that all periodizations with respect to the line assigned to a PE can be computed. This does not necessarily mean that every node has to store the whole data set, although the latter could be helpful in certain environments. The traditional row-column (RC) algorithm on the other hand requires each node to have access to only a subset of the rows or the columns of the original data array, but a severe communications overhead is introduced by the need to perform one or more global transpositions of the data.

Data entry to the multi-processor machine depends on the particular application in which the DFT is embedded. While in some applications the data are stored in the disk(s) and have to be imported to the nodes of the parallel machine, in other applications the data have already been imported during previous computational stages or have been generated locally in the nodes. Since the initial data loading is application dependent, we have not investigated the implementation of this step in detail. We have however considered two different models for the initial stage of the HRTA: In the first model, which is referred to as the *master-slaves* method, a master node computes all the periodizations and sends to the other nodes (the *slaves*) only the periodized data. Using this method the need for storage on the nodes is reduced since every one has to store only the periodized data. Furthermore the computation of the periodizations by the master node can be performed in a way that interleaves computation and communication steps in order to achieve optimum performance.

In the second model, also referred to as the *multi-processor* model, all nodes have access to *all* the data set, so that in an initial *loading phase*, either all nodes access a shared file system concurrently, or one node reads the data from a file and then *broadcasts* them to all the other nodes. Although the HRTA requires that larger (than for the RC method) data sets be sent to the nodes, the fact that these data sets are the same allows for the use of the broadcasting capabilities of the parallel machine. This approach is especially attractive for shared bus based machines where broadcasting can be performed efficiently. On the iPSC/860 hypercube broadcasting the whole

data set is faster than sending "chunks" of different data elements to each processor, but certainly much slower than an one-hop away communication step. After extensive experimentation with the iPSC/860 we concluded that the master-slaves model allows for more efficient implementations of the HRTA than the multi-processor model.

Since, for the hybrid algorithm, the node computations are completely independent, there is very little need for *synchronization* among the nodes. Therefore completely asynchronous implementations that exploit the MIMD nature of the machine and allow each node to perform its computations as soon as the data are received are possible. On the other hand, in the row-column method, that is the most commonly used method today, a series of distributed global data transpositions has to be performed, and its efficiency is highly dependent on the tight synchronization among the processors. Therefore, the increased need for communication during the loading phase that the hybrid algorithm has does not make it slower than the row-column method, unless more sophisticated methods for data distribution can be employed. (We intend to explore this issue in detail by investigating the capabilities, advantages and drawbacks of the CFS that the iPSC/860 supports).

C.5.3 Reporting the results to the host

The final phase of reporting results, as well as the initial phase of loading data depend on the DFT *application.* In some applications upon completion of the DFT the results need not be reported back to the host since they are further processed. In others, it is desired to store all the DFT values in the external disk memory. In the parallel HRTA we propose, the distribution of the results on the nodes is according to the lines they belong to. Whereas in some applications it is desired to organize the results in the same order as the original data, in others it is essential to return the results along subsets of the original index space (lines or planes) [10], [9]. Since the final reporting phase is highly dependent on the application, we have not investigated this issue in detail. We would like to mention however that the limited synchronization needs of the HRTA leads to flexible implementations of the final reporting phase, because the nodes can finish their computations independently and start returning their results asynchronously as soon as they become available.

C.6 Implementation Results

It has been a common belief among the signal processing community that with the pipelining and dual operations capabilities of the modern RISC microprocessors, there is no need for DFT algorithms for data sizes that are not a power of two. This was so because zero padding can be employed along with the highly optimized, microprocessor specific, power-of-two FFT routines. As we will show here, this is not true for multi-dimensional DFTs. Zero padding along many

dimensions can increase the data size tremendously and reduce the efficiency of the power-of-two routines drastically. Moreover, in a multiprocessor environment, the standard power-of-two Row-Column (RC) based FFT algorithms require one or more global transposition steps in which all processors need to communicate with every other processor in the network. Due to the limited bandwidth of the communication links, the global transposition steps result in a bottleneck that severely limits the maximum achievable speedup.

C.6.1 The 2D DFT case, $MP \times KP$

To demonstrate the advantages of the proposed hybrid RTA algorithm (HRTA) relative to the traditional row-column (RC) power-of-two algorithm, we compare an implementation for the 2D DFT case with a highly optimized Intel iPSC/860, vendor supplied RC implementation for the case $P = 3$, using $L = P + 1 = 4$ nodes. The HRTA periodization step was coded in Fortran, whereas for the 2D FFTs we used vendor supplied, assembly coded, power-of-two FFT routines, optimized for the i860 processor. Finally, for the 3-point DFTs step we also used optimized, hand coded in assembly, vectorized routines. We performed several tests for various non-power-of-two data sizes and we report the computational time achieved by both methods. The time is measured from the point that all the necessary data already reside in the nodes, and until the results have been computed and stored in the processors local memory. In both implementations the distribution of the results is different from the original data distribution. Using the HRTA the results are distributed along the lines assigned to each processor, and using the RC method the results are distributed in a transposed fashion.

In Table 1 we compare the speed of the two algorithms for a variety of data sizes. Depending on the amount of zero padding, the RC method could be up to about 70% slower than the HRTA. Moreover, our HRTA implementation can be further optimized (assembly coding of the periodization step), whereas the RC implementation is already fully optimized for the Intel iPSC/860. As we can see from Table 1, the speedup over the RC method increases with the data size as expected, since the amount of zero-padding increases with the size of the original non-power-of-two data set as well.

As an indication of the percentage of time spent on each one of the three major computational tasks we refer to the case: $M = 256$, $K = 256$, $P = 3$, (size 768×768). The times (in msec) for the computation of $MKP = 3 \cdot 2^{16}$ periodizations, $P = 3$ $M \times K = 256 \times 256$ 2D FFTs and $MK = 2^{16}$ 3-point DFTs respectively are: $t_p = 475.0208$, $t_{fft2d} = 542.0287$ and $t_{dft3p} = 97.6183$. As we can see, the time required for the periodizations almost equals that for the 2D FFTs. A careful assembly coding of the periodizations step is expected to reduce this time by at least 50%, thus making an optimized HRTA implementation twice as fast as the optimized RC implementation.

Hybrid Algorithm		Row-Column Method	
$MP \times KP$	time (msec)	size	time (msec)
192×192	66.7470	256×256	95.9750
192×384	130.8893	256×512	194.5686
384×384	254.7608	512×512	403.3229
384×768	511.9606	512×1024	866.7061
768×768	1117.2697	1024×1024	1876.8777

Table 1: Comparison of the performance of the HRTA parallel algorithm vs. the iPSC/860 optimized RC parallel algorithm implementation, for various data sizes, and $P = 3$. In both methods the data are assumed to initially reside in the nodes.

C.6.2 The 3D DFT case, $N \times MP \times KP$

We have implemented the parallel HRTA algorithm for the case $P = 3$, on $L = P + 1 = 4$ nodes, where N, M and K are assumed to be a power of two. In the 3D case, the periodization step can be organized to result in a much more regular memory access than in the 2D case, since now vector additions of data stored in consecutive memory locations can be employed. We coded this step using a mixture of Fortran and vector addition assembly routines, whereas assembly routines have been used for both the 3D FFTs and the 3-point DFTs.

In Table 2 we compare the HRTA with the optimized iPSC/860 implementation of the RC algorithm for a variety of data sizes. Using both methods the data initially reside in the nodes, and the time is measured up to the point that the results have been computed and stored in the local memory. As we can see from Table 2, the RC algorithm is on the average about 70% slower than the HRTA algorithm for a good mix of the cases tested. In the same table we also report the computational times required for the DFT of the same data set using the RC method on 8 nodes. It is interesting to observe that even if the number of nodes is doubled the performance is increased by only 15% on the average relative to the 4-node HRTA implementation.

In Figure 4 we compare our implementation of the parallel HRTA with the parallel RC method by plotting the (base 2) logarithm of the computational times required by both methods for data sizes $N \times 96 \times 96$, versus $\log N$. In the same figure we plot the ratio of the computational times ("speedup") as well. As we can see the RC method can be as much as 1.70 times slower than the HRTA for the range of N examined.

As an indication of the percentage of computational time spent in each stage of the HRTA, we report the times (in msec) required for the major tasks involved when the data size is $16 \times 192 \times 192$ (i.e. $N = 16$, $P = 3$, $M = K = 64$). In this case we need to compute: $NMKP = 3 \cdot 2^{16}$

Hybrid Algorithm		Row-Column Method		
$N \times MP \times KP$	time (msec, 4-PEs)	size	time1 (msec, 4-PEs)	time2(msec, 8-PEs)
$8 \times 96 \times 96$	183.6536	$8 \times 128 \times 128$	289.6318	150.3773
$8 \times 96 \times 192$	351.6283	$8 \times 128 \times 256$	592.9389	298.6226
$8 \times 192 \times 192$	719.3023	$8 \times 256 \times 256$	1278.7431	628.8135
$8 \times 192 \times 384$	1522.1512	$8 \times 256 \times 512$	2568.7333	1262.5482
$16 \times 96 \times 96$	338.0456	$16 \times 128 \times 128$	565.8212	273.1891
$16 \times 96 \times 192$	690.1413	$16 \times 128 \times 256$	1134.1762	556.7175
$16 \times 192 \times 192$	1422.3148	$16 \times 256 \times 256$	2245.3022	1175.6518
$4 \times 384 \times 384$	1791.1266	$4 \times 512 \times 512$	2941.7306	-

Table 2: Performance comparison of the 3D HRTA parallel algorithm vs. the iPSC/860 optimized RC parallel implementation, for a variety of data sizes and $P = 3$. In both methods the data are assumed to initially reside in the nodes. For the RC method we report both the 4-nodes and 8-nodes time.

periodizations, $P = 3$ $N \times M \times K = 16 \times 64 \times 64$ 3D FFTs, and $NMK = 2^{16}$ 3-point DFTs. The corresponding times are: $t_p = 302.2875$, $t_{fft3d} = 1022.5539$ and $t_{dft3p} = 97.7224$. It is interesting to notice that although the number of periodizations is the same $(3 \cdot 2^{16})$ as for the 2D DFT case (768×768) discussed in the previous subsection, t_p is reduced by about 35 %. This is because in the 3D DFT case, accesses to the data array are more localized than in the 2D case since periodizations are computed only along the two-dimensional planes. As we can see, the 3D FFTs computation is still the most expensive task. A 3D FFT $(16 \times 64 \times 64)$, although applied to a data set with the same number of elements as in the 2D case (256×256), is two times slower than a 2D FFT. This is mainly due to the fact that the 3D FFT requires more function calls to the optimized 1D FFT routine as well as additional transposition steps. The assembly coded 3-point DFT is again as fast as in the 2D case. The large percentage of the computational time that the 3D FFT requires makes us to believe that trying to limit the need for large 3D FFTs is more important than optimizing the periodizations.

In Table 3 we report execution times that include the initial data loading phase. In both implementations the data are assumed to initially reside in one node which then distributes them to all the others. For the hybrid method we used the master-slaves model, described in section 4, that works as follows: The master PE performs all the periodizations; as soon as one periodization is completed, the results are sent via non-blocking communications to a slave PE and the computation of the next periodization can start in the master PE. The slave node that receives the periodized data can proceed with the 3D FFTs. This interleaving between computations and communications

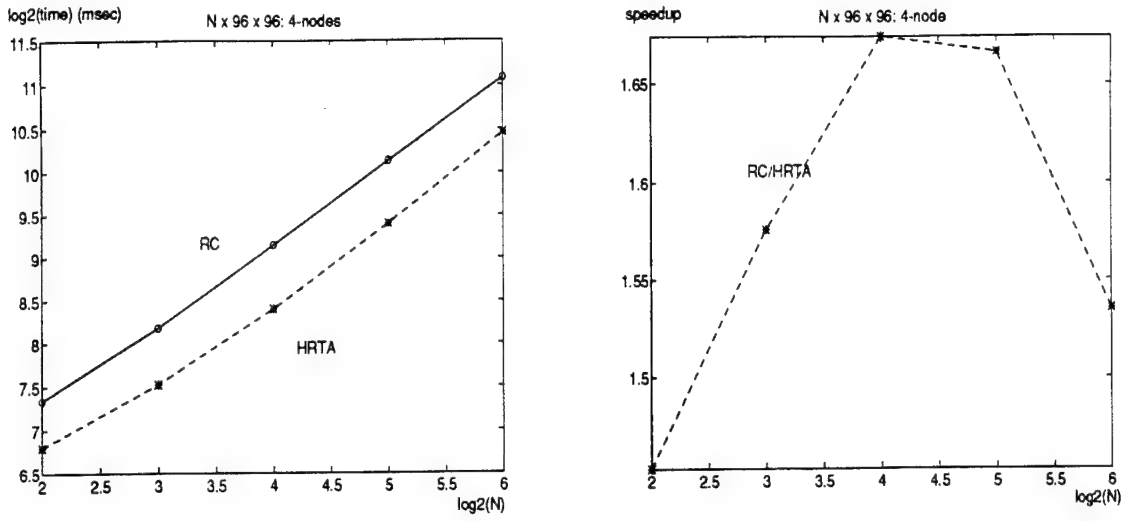


Figure 4: Performance comparison of the 4-node 3D HRTA parallel algorithm vs. the 3D RCA method. *Left:* plots of the (base 2) log. of the computational time (in milliseconds) versus $\log N$. For the HRTA the data sizes used were of the form $N \times 96 \times 96$ and for the RC method the corresponding sizes were zero padded to $N \times 128 \times 128$. *Right:* the ratio of times T_{rc}/T_{hrta} (speedup)

achieves optimum performance using the HRTA. In the row-column method each one of the four PEs needs only $\frac{1}{4}$ of the data set. Including the data loading phase leads to even larger improvements over the RC method. This is due to the asynchronous nature of the hybrid method implementation that allows data loading in a pipelined fashion to further reduce the total DFT time.

The final reporting of the results to the master node, can also be done in pipelined fashion. The nodes do not finish their computations all at the same time. The master node finishes first; it can then re-shuffle its own data back into order and then receive messages from the other nodes. As soon as each node finishes its computation, it can return its part of the results to the master node. On the other hand, in the RC method all nodes finish almost simultaneously and the total reporting time will be the sum of the times required by each individual node to return its results to the master node. As we can see from Table 3 (column labeled *time2*) when the final reporting stage is included the advantage of the HRTA becomes even greater.

C.6.3 The hybrid algorithm implementation for larger sizes of P

In this subsection we present preliminary results on the performance of the HRTA implementations for 3D DFTs of sizes $N \times MP \times KP$, where the prime number is $P = 5$ or $P = 7$. In Tables 4 and 5 we report execution times in six and eight nodes respectively.

In Figure 5 we plot the computational time versus the size of the problem as well as the

Hybrid Algorithm (4 nodes)			Row-Column Method (4 nodes)		
$N \times MP \times KP$	time1 (msec)	time2 (msec)	size	time1 (msec)	time2 (msec)
$64 \times 48 \times 6$	121.89	146.97	$64 \times 64 \times 8$	148.27 (+21.64%)	229.92 (+56.44 %)
$64 \times 96 \times 6$	210.28	290.53	$64 \times 128 \times 8$	291.01 (+38.39%)	451.78 (+55.50%)
$128 \times 96 \times 6$	477.11	573.18	$128 \times 128 \times 8$	587.26 (+23.08%)	905.37 (+57.95%)
$128 \times 192 \times 6$	940.17	1136.35	$128 \times 256 \times 8$	1214.48 (+29.17%)	1882.26 (+65.64%)
$16 \times 96 \times 96$	791.75	1007.59	$16 \times 128 \times 128$	1139.49 (+43.92%)	1764.86 (+75.16%)

Table 3: Comparison of the performance of the 3D HRTA vs. the RC method. The data initially reside in one master node; *time1* includes the data distribution whereas *time2* includes in addition the final reporting to the master node. For the RC method, the percentages in parenthesis correspond to: $100 \cdot \frac{(T_{rc}-T_{hrta})}{T_{hrta}}$

Hybrid Algorithm (6-nodes)		Row-Column Method (8-nodes)	
$N \times MP \times KP$	time (msec)	size	time (msec)
8 x 160 x 320	745.4254	8 x 256 x 512	1262.5699
8 x 160 x 160	388.6131	8 x 256 x 256	628.7040
16 x 80 x 160	395.3691	16 x 128 x 256	556.5085
32 x 80 x 80	392.0486	32 x 128 x 128	587.5301
64 x 40 x 80	395.1672	64 x 64 x 128	577.1927
128 x 40 x 40	414.7549	128 x 64 x 64	578.8189
2048 x 10 x 10	856.1560	2048 x 16 x 16	714.3222

Table 4: Comparison of the hybrid algorithm implementation and the row-column method, for $P = 5$.

speedup ratio over the RC method applied to a data set zero padded up to the next power of two in each dimension. As we can see from Figure 5 although the optimized RC algorithm runs on 8-nodes, instead of 6 for the HRTA algorithm, it is about 1.5 times slower than the non-optimized HRTA implementation.

C.6.4 The node clustering approach

As we have seen earlier in the four-node 3D DFT case, each node needs to perform three 3D FFTs of size $M \times K \times N$. If the size of the 3D FFT data is too large to fit into a single node, or faster implementations are desired, the four nodes can now be considered as four conceptual *clusters* of nodes. In each of the clusters, the 3D data is distributed along the first dimension, and both the periodization and 3-point DFT steps can be performed independently by each node of a cluster.

Hybrid Algorithm (8-nodes)		Row-Column Method (8-nodes)	
$N \times MP \times KP$	time (msec)	size	time (msec)
8 x 224 x 224	647.1830	8 x 256 x 256	628.7040
16 x 112 x 224	595.7953	16 x 128 x 256	556.5085
32 x 112 x 112	584.6240	32 x 128 x 128	587.5301
64 x 56 x 112	567.5866	64 x 64 x 128	577.1927
128 x 56 x 56	577.8302	128 x 64 x 64	578.8189
2048 x 14 x 14	1109.0271	2048 x 16 x 16	714.3222

Table 5: Comparison of the hybrid algorithm implementation and the row-column method, for $P = 7$.

For the 3D FFT computation, only communication among the processors of the same cluster is needed, thus greatly reducing the total communication requirements. In Figure 6 we show how an eight-node hypercube is organized in 4 clusters to compute the 3D DFT of size $N \times 3M \times 3K$.

Node clustering can be used to create *scalable* implementations that make full utilization of the available hardware. If the number of nodes is 2^n and four clusters are used, every node needs to store only the $\frac{1}{2^{n-2}}$ of the original data set. In Tables 6, 7 and 8 we present timing results for 8, 16, and 32 nodes 4-cluster implementations ($P = 3$) and compare the performance of the HRTA with that of the highly optimized row-column method using zero-padding up to the next power of two in every dimension. It is again assumed that the data already reside in the nodes before the processing starts. Moreover, the three 3D FFTs computed by every cluster are implemented using the optimized row-column routines.

Hybrid Algorithm		Row-Column Method	
$N \times MP \times KP$	time (msec)	size	time (msec)
$16 \times 192 \times 192$	1027.9498	$16 \times 256 \times 256$	1175.2196
$8 \times 384 \times 192$	1052.3800	$8 \times 512 \times 256$	1330.3198
$8 \times 384 \times 384$	2206.7041	$8 \times 512 \times 512$	2669.4727

Table 6: HRTA in 8-nodes = 4 clusters of 2 PEs/cluster vs. 8-nodes RC with zero padding

In Figure 7, we plot the computational time required by each implementation, versus $\log N$ for data sizes of the form $N \times 96 \times 96$. In the same figure we also show the ratio T_{rc}/T_{hrta} as before. As we can see from Figure 7, the hybrid algorithm is only slightly better than the row-column method. However, the periodization part of our code is just in standard Fortran implementation and it can be further optimized.

In Table 7 we present timing results for a 16-node implementation; $P + 1 = 4$ clusters (with

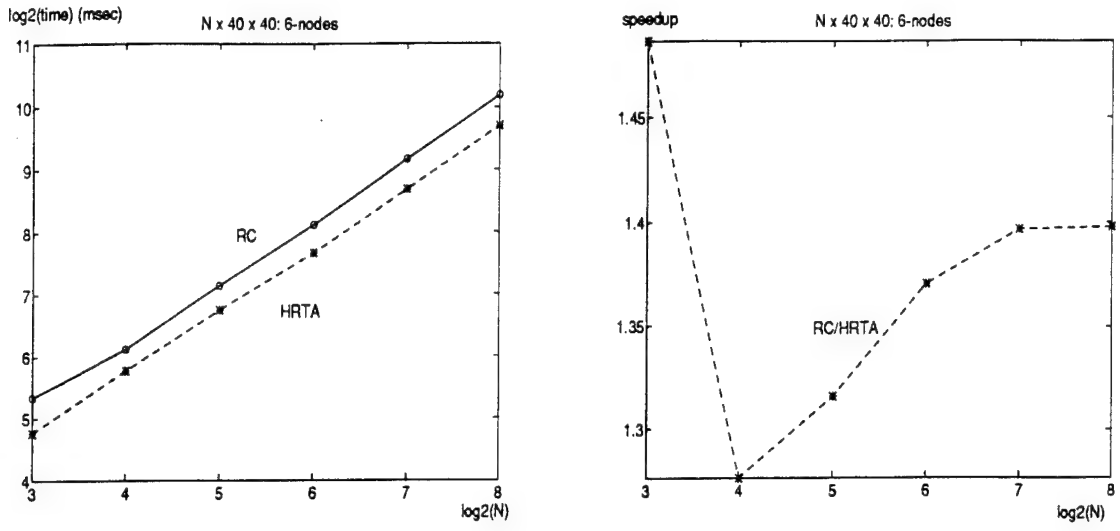


Figure 5: Comparison of the 6-node HRTA to the 8-node RC implementation: *Left:* Plots of the (base 2) logarithm of the computational time (in msec) vs. $\log N$. The data sizes used with the HRTA were of the form $N \times 40 \times 40$, $P = 5$. The corresponding RCA data sizes were of the form $N \times 64 \times 64$. *Right:* the speedup ratio T_{rc}/T_{hrta}.

4 PEs/cluster) cooperate to perform the 3D DFT. Each of the cluster has the whole data set stored in it. Within a cluster, each of the nodes stores 1/4 of the data (distributed along the first dimension). Three four-node row-column 3D FFTs are performed within each cluster. In Figure 8 we show the data distribution within one of the clusters. As we can see from Table 7, the non-optimized HRTA implementation has comparable performance with the optimized RC implementation. The computational time versus the size of the data set and the speedup ratio over the row-column method is shown in Figure 9. Finally, in Table 8, we present timing results for a 32-node HRTA implementation, partitioned into 4 clusters with 8 PEs each.

The node clustering approach can be used in general for any size of the prime number P. As an

Hybrid Algorithm		Row-Column Method	
$N \times MP \times KP$	time (msec)	size	time (msec)
$16 \times 192 \times 192$	544.8489	$16 \times 256 \times 256$	569.4243
$16 \times 384 \times 192$	1103.3147	$16 \times 512 \times 256$	1204.5177
$16 \times 384 \times 384$	2289.6839	$16 \times 512 \times 512$	2411.0360
$8 \times 384 \times 768$	2373.4150	$8 \times 512 \times 1024$	-

Table 7: HRTA in 16-nodes = 4 clusters of 4 PEs/cluster vs. 16-nodes RC with zero padding.

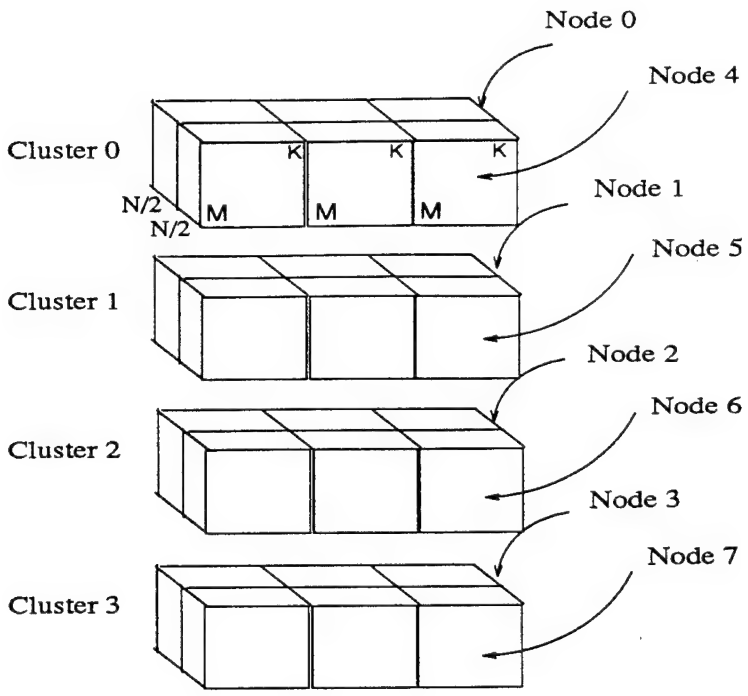


Figure 6: An 8-node hypercube, organized into $P + 1 = 4$ clusters of 2 nodes each. Only near-neighbor communications inside every cluster are need to compute an $N \times MP \times KP$ 3D FFT.

example, we have implemented the case $N \times 5M \times 5K$ ($P = 5$) in a 12-node configuration. The 12 nodes are partitioned into $6 = P + 1$ clusters, each one having two nodes. The data are distributed evenly within each cluster along the first dimension, and the row-column 3D FFT kernel is used to perform the 3D DFTs inside every cluster. Only communication among PEs in the cluster are needed. In Table 9 we compare the HRTA implementation versus the RC method running running on 16 nodes. As we can see the 12-node HRTA outperforms the 16-node RC implementation.

Hybrid Algorithm		Row-Column Method	
$N \times MP \times KP$	time (msec)	size	time (msec)
$128 \times 96 \times 96$	532.9717	$128 \times 128 \times 128$	589.2343
$64 \times 192 \times 192$	1039.3567	$64 \times 256 \times 256$	1173.0707
$32 \times 384 \times 192$	1071.3325	$32 \times 512 \times 256$	1216.7244
$32 \times 384 \times 384$	2208.9439	$32 \times 512 \times 512$	2444.6264

Table 8: HRTA in 32-nodes = 4 clusters of 8 PEs/cluster vs. 32-nodes RC with zero padding.

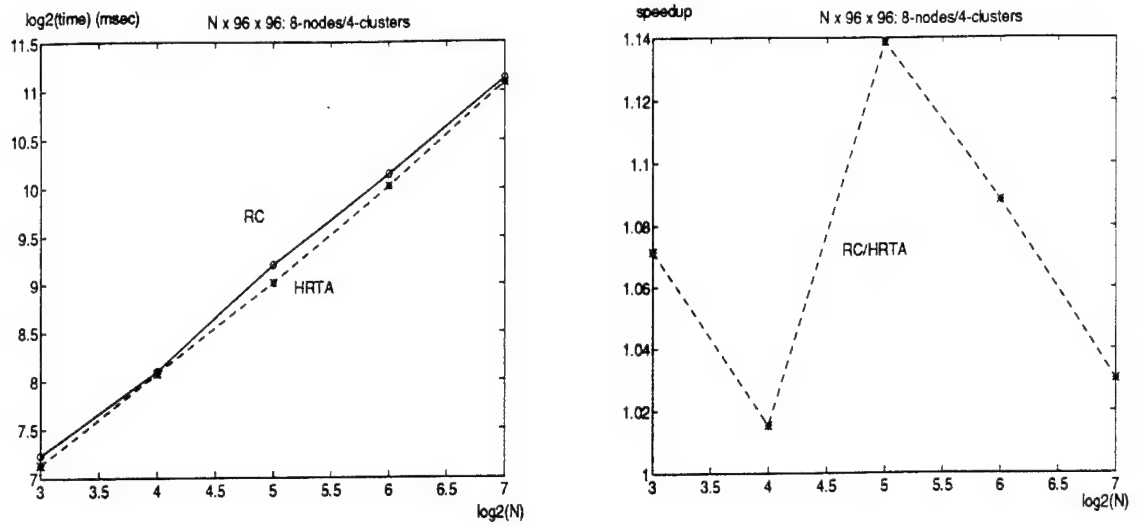


Figure 7: Performance comparison of the 8-node HRTA (4 clusters with 2 PEs/cluster) vs. the RC method. *Left:* Plots of the (base 2) logarithm of the computational time (in milliseconds) vs. $\log N$. *Right:* the speedup ratio Trc/T_{hrta}.

Hybrid Algorithm (12-nodes)		Row-Column Method (16-nodes)	
$N \times MP \times KP$	time (msec)	size	time (msec)
$16 \times 80 \times 80$	137.7877	$16 \times 128 \times 128$	139.8977
$16 \times 160 \times 80$	260.3311	$16 \times 256 \times 128$	279.9925
$16 \times 160 \times 160$	512.0763	$16 \times 256 \times 256$	567.8107
$16 \times 320 \times 160$	979.6029	$16 \times 512 \times 256$	1201.9575

Table 9: Comparison between a 12-node HRTA parallel algorithm with clustering (6 clusters, 2 PEs/cluster), and the RC method running on 16 nodes. The HRTA is faster although it uses 25 % less nodes.

C.6.5 Conclusions and further Research directions

A new approach for computing multi-dimensional DFTs with limited interprocessor communications has been proposed, and its advantages relative to the standard row-column power-of-two based FFT algorithms has been demonstrated. Although it has been a common belief that with the available modern RISC microprocessors there is no need for new "exotic" DFT algorithms, we have shown that substantial computational savings can be achieved in a parallel environment by using a more flexible hybrid scheme. The DFT is a major component of numerous signal and image processing applications and if real-time operation is envisioned, only parallel processing can satisfy the user demands.

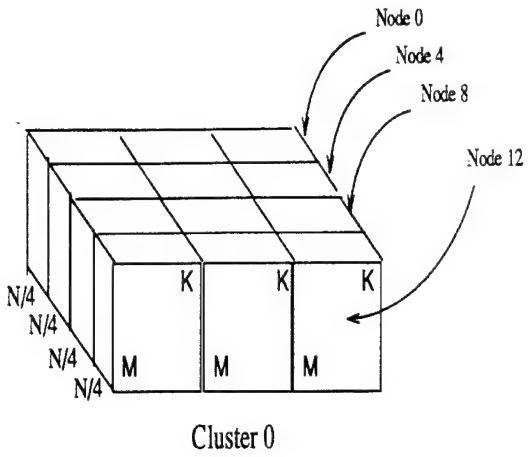


Figure 8: A 4-node cluster, part of a 16-node, 4-clusters hypercube: Only communications inside each cluster need to be performed to compute a 3D FFT.

The proposed hybrid algorithm combines the advantages of both the recently proposed RTA and the Cooley-Tuckey RC method to give optimal parallel realizations for non-power-of-two data sizes. We demonstrated the flexibility and the efficiency of the HRTA by implementing it on an Intel iPSC/860 hypercube, where our non-optimized HRTA realizations outperform the highly optimized RC method realizations. The HRTA provides an alternative that is suitable for many different parallel and distributed processing environments. In a DSP board with 4 compute nodes communicating via a shared bus, the HRTA seems to be the only viable parallel processing scheme that would achieve real-time performance. In Clusters Of WorkstationS (COWS), a rapidly emerging cost-effective model for parallel computing, the need for an all-to-all communication, that is necessary for transposition using the RC method, would render the RC method highly inefficient. On the other hand, the HRTA has little or no need for communication between different workstations so that very fast implementations can be created.

We have demonstrated that the HRTA shares the scalability properties of the RC algorithm so that multi-dimensional DFTs of large data sizes can be computed efficiently on parallel architectures. To optimize the HRTA the periodization step can be further improved. The modulo arithmetic based addressing can be avoided if more local memory is allocated to store two integer arrays Ind1(M,P) and Ind2(K,P) used as index-lookup tables. Their entries can be either computed once or preloaded along with the data. An alternative approach is to replace the modulo operations with additions and conditional statements. Moreover, since along the index n, the periodizations reflect essentially to vector additions, efficient assembly language modules that make full use of the pipelining capabilities of the i860's RISC architecture can be employed. Therefore, the larger the N the better the use of the CPU pipelining capabilities and of the cache memory. Along the other

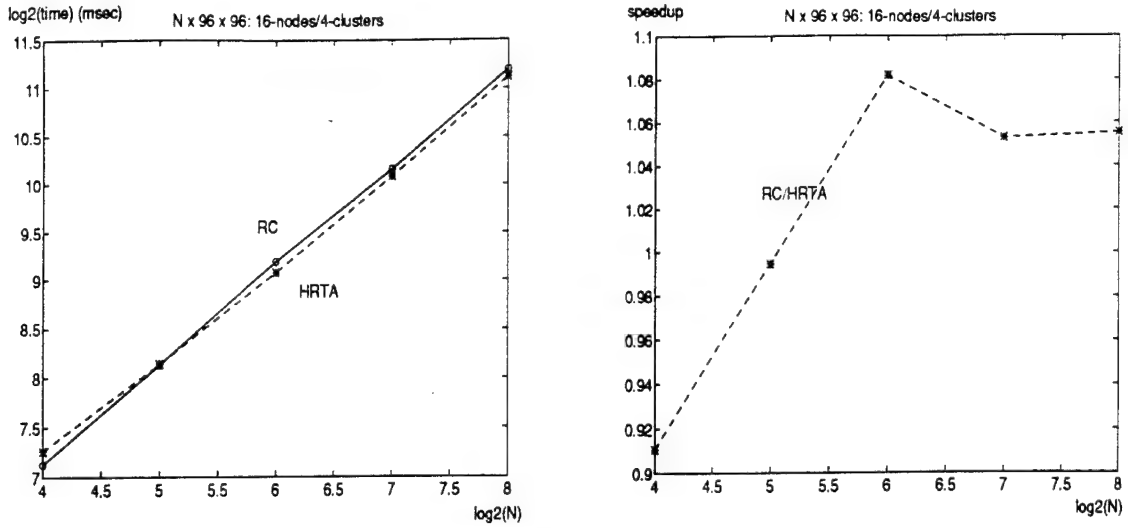


Figure 9: Performance comparison of the 16-node (4 clusters with 4 PEs/cluster) HRTA vs. the 16-nodes RC method. *Left:* Plots of the (base 2) logarithm of the computational time (in msec) vs. $\log N$. The data sizes used are of the form $N \times 96 \times 96$, zero padded to $N \times 128 \times 128$ for the RC method. *Right:* the speedup ratio T_{rc}/T_{hrta}

indices, the memory addresses to be referenced do not follow a sequential pattern, so that extra care must be taken to prefetch the necessary data before they are needed.

Also note that there is a lot of flexibility on how the nested loops can be arranged in order to compute the periodized data g_l from the original data set f. This flexibility can also be used to minimize cache misses. For a large prime number P, the ordering of the nested loops that sequentially addresses the elements of the larger data array f, seems more advantageous. This is because the ratio of sizes of the two arrays f and g_l increases with P, and the whole g_l matrix can most probably fit into the cache. Therefore accessing the data array f sequentially allows to reduce the cache misses since the data f can be imported into the cache in a column by column fashion and then transformed to the periodized data.

The execution of the P multi-dimensional FFTs could become faster by either grouping together or interleaving the tasks involved. Recall that each one of the 2D or 3D parallel FFTs using clustering consists of three tasks: (i) 2D or 1D FFTs, (ii) Communication (global transposition) and (iii) 1D FFTs. Therefore the following two optimizations are possible: (1) Group all the corresponding tasks of the P multi-dimensional FFTs together and do the same with the communication stages, so that *only one* communication startup time is needed instead of P. (2) Employ a vector-pipelined parallel 3D FFT: Using *asynchronous* communication calls, computations associated with the next FFT can be interleaved with communications required for the previous FFT. Since the

communication time accounts for as much as 50 % of the overall time, pipelining strategies are expected to greatly improve the performance.

When each one the nodes has completed the computation of a set of multi-dimensional FFTs, the independent P-point DFTs have to be performed. One method is to compute the P-point DFTs *after* all the previous computations associated with the data set have been completed. It has the advantage of allowing the use of highly optimized vectorized assembly routines. Another method is to use the partial contribution of each sub-block of data after the partial periodization and 2D (3D) FFT is being computed and as soon as these data become available. Note that as soon as a 2D (3D) FFT for a part of the periodized data, corresponding to a point on the line, has been computed the contribution of that buffer to the overall P-point DFT can be computed. Although this implementation has the drawback of requiring a larger amount of computations relative to the first method, it has the advantage of more efficient balancing between computations and communications since these two phases can be interleaved.

The extension of the HRTA algorithm to include periodizations with respect to sets of higher dimensionality (planes instead of lines) is worth investigating. It is expected to lead to completely asynchronous implementations that take full advantage of the large number of processors available and increase the achievable efficiency for problems of very large size.

C.6.6 Acknowledgements

We would like to thank the Intel Supercomputing Center, Oregon for their assistance and advice and for providing us the iPSC/860 hypercube to develop our programs. We would also like to thank Dr. S. Qian for providing us with the efficient vectorized P-point DFT assembly codes and for the fruitful discussions and his assistance for the completion of the project.

References

[1] R. Tolimieri, M. An, and C. Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, New York, 1989.

[2] Auslander L., Feig E., and S. Winograd. New Algorithms for the Multidimensional Discrete Fourier Transform. *IEEE Transactions on ASSP*, 31:388–403, February 1983.

[3] I. Gertner. A New Efficient Algorithm to compute the Two-Dimensional Discrete Fourier Transform. *IEEE Transactions on ASSP*, 36:1036–1050, July 1988.

[4] H.J. Nussbaumer and P. Qualdalle. Fast Computation of Discrete Fourier Transforms using Polynomial Transforms. *IEEE Transactions on ASSP*, 27:169–181, 1979.

[5] M. Vulis. The Weighted Redundancy Transform. *IEEE Transactions on ASSP*, 37:1687–1692, November 1989.

[6] M. An, I. Gertner, M. Rofhear, and R. Tolimieri. Discrete Fast-Fourier Transforms: A Tutorial Survey. In *Advances in Electronics and Electron Physics*, volume 80, pages 2–69. Academic Press, 1991.

[7] R. Tolimieri, M. An, and C. Lu. *Mathematics of Multidimensional Fourier Transform Algoritms.* Springer-Verlag, New York, 1993.

[8] M. Rofheart. *Algorithms and Methods for Multidimensional Digital Signal Processing.* PhD thesis, CUNY, 1991.

[9] A. Kojima, N. Sakurai, and J. Kishigami. Motion Detection using 3D-FFT Spectrum. In *ICASSP Int. Conf. Accoustics, Speech and Signal Proc.*, volume 5, pages 213–216, 1993.

[10] Boaz Porat and Benjamin Friedlander. A Frequency Domain Algorithm for Multiframe Detection and Estimation of Dim Targets. *IEEE Trans. on PAMI*, 12:398–401, April 1990.

[11] I.J. Good. The Interaction Algorithm and Practical Fourier Analysis. *J. Roy. Stat. Soc. Ser. B*, 20:361–372, 1958.

[12] L.H. Thomas. Using a Computer to Solve Problems in Physics. In *Applications of Digital Computers.* Ginn and Co., Boston, MA, 1963.

[13] S. Lang. *Algebra.* Addison Wesley Publising Company, 1970.

[14] C.S. Burrus and P.W. Eschenbacher. An In-Place In-Order Prime Factor FFT Algorithm. *IEEE Transactions on ASSP*, 29:806–817, 1981.

[15] D.P. Kolbe and T.W. Parks. A Prime Factor FFT Algorithms Using High-Speed Convolution. *IEEE Transactions on ASSP*, 25, 1977.

[16] C. Temperton. A new set of Minimum-add Small-n Rotated DFT Modules. *J. of Comput. Physics, to appear*, 1993.

[17] S. Chu and C.S. Burrus. A Prime Factor FFT Algorithm using Distributed Arithmetic. *IEEE Transactions on ASSP*, 30:217–227, February 1982.

[18] c. Van Loan. Computational Frameworks for the Fast Fourier Transform. In *SIAM Frontiers in Applied Mathematics*, Philadelphia, 1992.

[19] D. S. Scott. Parallel I/O and Solving Out of Core Systems of Linear Equations. In *Proc. DAGS'93 Symposium*, pages 125–130, Hanover, NH, 1993.

D Weyl-Heisenberg Systems and the Finite Zak Transform

Abstract

Previously, a theoretical foundation for designing algorithms for computing Weyl-Heisenberg coefficients at critical sampling was established applying the finite Zak transform. This theory established clear and easily computable conditions for existence of Weyl-Heisenberg expansion and for stability of computations. The main computational task in the resulting algorithm was a 2-dimensional finite Fourier transform.

In this work we extend the applicability of the approach to rationally oversampled Weyl-Heisenberg systems by developing a deeper understanding of the relationship established by the finite Zak transform between linear algebra properties of Weyl-Heisenberg systems and function theory in Zak space. This relationship will impact on questions of existence, parameterization and computation of Weyl-Heisenberg expansions.

Implementation results on single RISC processor of i860 and the PARAGON parallel multiprocessor system are given. The algorithms described in this paper possess highly parallel structure and are especially suited in a distributed memory parallel processing environment. Timing results show that real-time computation of W-H expansions is realizable.

D.1 Introduction

During the last four years powerful new methods have been introduced for analyzing Wigner transforms of discrete and periodic signals [7, 8, 10] based on finite Weyl-Heisenberg (W-H) expansions [1, 4, 5, 9]. A recent work [7] adapted these methods to gain control over the cross-term interference problem [6] by constructing signal systems in time frequency space for expanding Wigner transforms from W-H systems based on Gaussian-like signals.

The computational feasibility of the method in [7] depends strongly on the availability of efficient and stable algorithms for computing W-H expansion coefficients. Since in general, W-H systems are not orthogonal, standard Hilbert space inner product methods do not apply. Moreover since critically sampled W-H systems may not form a basis, oversampling in time-frequency is necessary for the existence of arbitrary signal expansions. In fact this is usually the case for systems based on the Gaussian. In [7, 8, 9, 10, 11], the concept of biorthogonals was applied to the problem of W-H coefficient computation. In [11], the Zak transform provided

the framework for computing biorthogonals for rationally oversampled W-H systems forming frames. A similar approach for critically and integer oversampled W-H systems can be found in [2, 3]. The goal in this work is somewhat different in that major emphasis is placed on describing linear spans of W-H systems which are not necessarily complete and on establishing in a form suitable for RISC and parallel processing, algorithms for computing W-H coefficients of signals in such linear spans. For the most part our approach extends on that developed in [2] and frame theory, an important part in [11] plays no role in this work. However as in these previous works, the finite Zak transform will be established as a fundamental and powerful tool for studying critically sampled and rationally oversampled W-H systems and for designing algorithms for computing W-H coefficients for discrete and periodic signals. The role of the finite Zak transform is analogous to that played by the Fourier transform in replacing complex convolution computations by simple pointwise multiplication. In this new setting properties of W-H systems such as their spanning space and dimension can be determined by simple operations on functions in Zak space. This relationship will impact on questions of existence, parameterization and computation of W-H expansions.

In the oversampled case both integer and rational oversampling are investigated. Implementation results on single RISC processor of i860 and the PARAGON parallel multiprocessor system are given for sample sizes both of powers of 2 and mixed sizes with factors 2, 3, 4, 5, 6, 7, 8, 9. The algorithms described in this paper possess highly parallel structure and are especially suited in a distributed memory parallel processing environment. Timing results on single i860 processor and on 4- and 8-node computing systems show that real-time computation of W-H expansions is realizable.

In section 2, the basic preliminaries will be established. Algorithms will be described in section 3 for critically sampled W-H systems, in section 4 for integer oversampled systems and in section 5 for rationally oversampled systems. Implementation results will be given in sections 6, 7 and 8.

D.2 Preliminaries

D.2.1 Weyl-Heisenberg systems

Choose an integer $N > 0$. A discrete function $f(a)$, $a \in Z$ is called N-periodic if

$$f(a + N) = f(a), \quad a \in \mathbf{Z}.$$

Denote by $L(N)$ the Hilbert space of all N-periodic functions with inner product

$$< f, g >= \sum_{a=0}^{N-1} f(a)g^*(a), \quad f, g \in L(N)$$

For $0 \leq m, n < N$ and $g \in L(N)$ define $g_{m,n} \in L(N)$ by

$$g_{m,n}(a) = g(a+m)e^{-2\pi ina/N}, \quad a \in \mathbf{Z}. \tag{1}$$

Suppose $N = KM = K'M'$. The *Weyl-Heisenberg* (W-H) *System* (g, M', K) is the set of functions

$$\{g_{m'M',n'K} : \ 0 \leq m' < K', \ 0 \leq n' < M\}. \tag{2}$$

We distinguish three cases

critically sampled $K = K', M = M',$
oversampled $K' > K, \ M' < M,$

We further distinguish two classes of oversampled W-H systems.

Integer oversampled $M = RM', \ R \in \mathbf{Z}$
Rational oversampled $M = RM', \ R \in \mathbf{Q}, \ R \notin \mathbf{Z}.$

undersampled $K' < K, \ M' > M.$

An expansion of $f \in L(N)$ over a W-H system is called a W-H *expansion.*

D.2.2 Finite Zak transform (FZT)

Suppose $N = KM$. For $f \in L(N)$ define the *finite Zak Transform* (FZT), $Z(K)f(a,b), \ a, b \in \mathbf{Z}$ by

$$Z(K)f(a,b) = \sum_{r=0}^{K-1} f(a+Mr)e^{2\pi ibr/K}, \quad a, b \in \mathbf{Z}. \tag{3}$$

Elementary properties of FZT including FZT based algorithms for computing W-H expansions over complete critically sampled W-H systems can be found in [2]. We will briefly discuss these results without proof and extend the role of the FZT to general W-H systems.

Theorem 1 *If $f \in L(N)$ then*

$$Z(K)f(a+M,b) = e^{-2\pi ib/K} Z(K)f(a,b), \quad a,b \in \mathbf{Z}. \tag{4}$$

$$Z(K)f(a,b+K) = Z(K)f(a,b), \quad a,b \in \mathbf{Z}. \tag{5}$$

Theorem 1 implies $Z(K)f$ is N-periodic in each variable and is completely determined by its values

$$Z(K)f(a,b), \quad 0 \le a < M, \ \ 0 \le b < K. \tag{6}$$

Denote by $L(M,K)$ the Hilbert space of all functions $F(a,b)$, $0 \le a < M$, $0 \le b < K$, with inner product

$$< F,G >= \sum_{a=0}^{M-1} \sum_{b=0}^{K-1} F(a,b)G^*(a,b), \quad F,G \in L(M,K). \tag{7}$$

Define $Z_0(K)f \in L(M,K)$ by

$$Z_0(K)f(a,b) = Z(K)f(a,b), \quad 0 \le a < M, \ \ 0 \le b < K. \tag{8}$$

In [2] we find the following theorem.

Theorem 2 *The mapping $K^{-1/2}Z_0(K)$ is an isometry from $L(N)$ onto $L(M,K)$. If $F \in L(M,K)$ and $f \in L(N)$ is defined by*

$$f(a+Mr) = K^{-1} \sum_{b=0}^{K-1} F(a,b)e^{-2\pi ibr/K}, \quad 0 \le a < M, \ \ 0 \le b < K, \tag{9}$$

Then $F = Z_0(K)f$.

For $f = L(N)$ and $F = Z_0(K)f$, we can summarize the preceding discussion by the matrix formula

$$\begin{bmatrix} F(0,0) & F(1,0) & \cdot & \cdot & F(M-1,0) \\ F(0,1) & & & & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ F(0,K-1) & \cdot & & \cdot & F(M-1,K-1) \end{bmatrix}$$

$$= F(K) \begin{bmatrix} f(0) & f(1) & \cdot & \cdot & f(M-1) \\ F(M) & & & & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ f((K-1)M) & \cdot & & \cdot & f(N-1) \end{bmatrix},$$

where $F(K)$ is the K-point Fourier transform matrix

$$F(K) = \left[w^{jk}\right]_{0\le j,k<K}, \quad w = e^{2\pi i\frac{1}{K}}.$$

Throughout this work we will identify $L(N)$ with $L(M,K)$ by theorem 2 and the matrix formula. For the most part, including the computation of W-H expansions, once we are in $L(M,K)$ we never need to formally return to $L(N)$.

D.2.3 Basic formulas

The following two theorems are proved in [2].

Theorem 3 *If $g \in L(N)$, $N = KM$, and $0 \le m,n < N$, then*

$$Z(K)g_{m,n}(a,b) = e^{-2\pi ian/N} Z(K)g(a+m, b-n), \quad a,b \in \mathbf{Z}. \tag{10}$$

In particular, if $0 \le m' < K$, $0 \le n' < M$, then

$$Z(K)g_{m'M,n'K}(a,b) = Z(K)g(a,b)e^{-2\pi i(n'a/M + m'b/K)}, \quad a,b \in \mathbf{Z}. \tag{11}$$

By theorem 1, the product function

$$Z(K)f(a,b)Z^*(K)g(a,b), \quad a,b \in \mathbf{Z} \;\; f,g \in L(N), \tag{12}$$

is M-periodic in the variable a and K-periodic in the variable b and can be viewed as a function in $L(M,K)$. The Fourier expansion of the product function is given in the following theorem.

Theorem 4 *For $f,g \in L(N)$, $N = KM$,*

$$Z(K)f(a,b)Z^*(K)g(a,b) = \frac{1}{M}\sum_{m'=0}^{K-1}\sum_{n'=0}^{M-1} < f, g_{m'M,n'K} > e^{-2\pi i(n'a/M + m'b/K)} \tag{13}$$

D.3 Critically Sampled W-H Systems.

Theorem 4 is a powerful tool for analyzing W-H systems. We first consider critically sampled W-H systems by extending the following result [2].

Theorem 5 *The critically sampled W-H system*

$$(g,M,K) = \{g_{m'M,n'K} \;\; 0 \le m' < K, \;\; 0 \le n' < M\} \tag{14}$$

is a basis of $L(N)$ if and only if $G = Z_0(K)g$ never vanishes.

By theorem 4 and the linear isomorphism established in theorem 2, we can identify the space of all $f \in L(N)$ satisfying

$$< f, g_{m'M,n'K} >= 0, \quad 0 \leq m' < K, \quad 0 \leq n' < M, \tag{15}$$

with the space of all $F \in L(M,K)$ satisfying

$$FG = 0, \quad G = Z_0(K)g. \tag{16}$$

The space of such $F \in L(M,K)$ can be identified with the orthogonal complement of the linear span of (g,M,K). If G never vanishes this complement is $\{0\}$ and (g,M,K) is a basis of $L(N)$ which is the content of theorem 5. More generally, we have the following result.

Theorem 6 *If the zero set ζ of $G = Z_0(K)g$ has exactly J points then the dimension of the linear span of (g,M,K) is $N-J$. A function $f \in L(N)$ is in the linear span of (g,M,K) if and only if $F = Z_0(K)f$ vanishes on ζ.*

If F vanishes on ζ, then we can write

$$F = GP, \quad P \in L(M,K).$$

In this case

$$f = \sum_{m'=0}^{K-1} \sum_{n'=0}^{M-1} c(m'M, n'K) g_{m'M,n'K} \tag{17}$$

if and only if

$$P(a,b) = \sum_{m'=0}^{K-1} \sum_{n'=0}^{M-1} c(m'M, n'K) e^{-2\pi i(an'/M + bm'/K)}, \tag{18}$$

The W-H expansion coefficients of f over (g,M,K) are given by the 2D $M \times K$ FT of P.

If G never vanishes then P is uniquely determined and the mapping

$$P \leftrightarrow F = GP \leftrightarrow f, \quad F = Z_0(K)f$$

defines a linear isomorphism from $L(M,K)$ onto $L(N)$.

Suppose that the zero set ζ of G has exactly J points with $J > 0$. Then (g,M,K) is linearly dependent and does not span $L(N)$. Choose $f \in L(N)$ in the linear span of (g,M,K). For each function

$$\alpha : \zeta \to \mathbf{C}$$

define $P = P^\alpha \in L(M,K)$ by

$$P(a,b) = \begin{cases} \frac{F(a,b)}{G(a,b)}, & (a,b) \notin \zeta, \\ \alpha(a,b) & (a,b) \in \zeta. \end{cases} \tag{19}$$

The space of such P is a J-dimensional subspace of $L(M,K)$. Since F vanishes on ζ, $F = GP$ leading to the next result.

Theorem 7 *If the zero set ζ of $G = Z_0(K)g$ has exactly J points then every f in the linear span of (g,M,K) has a J-dimensional space of W-H expansions over (g,M,K). The coefficient space of W-H expansions of f over (g,M,K) is given by the set of all 2D $M \times K$ FT of the J-dimensional space of functions $P^\alpha \in L(M,K)$.*

D.4 Integer Oversampled W-H Systems

Suppose $N = MK = M'K'$ with $M = RM'$, $R \in \mathbf{Z}$. The integer *oversampled* W-H system $\mathbf{g} = (g, M', K)$ is the disjoint union of critically sampled W-H systems.

$$\mathbf{g} = \bigcup_{r=0}^{R-1} \mathbf{g}_r, \quad \mathbf{g}_r = (g_r, M, K), \quad g_r = g_{rM',0}, \quad 0 \le r < R. \tag{20}$$

It is just as simple to consider the more general case where $\mathbf{g}$ is the disjoint union of critically sampled W-H systems $\mathbf{g}_r = (g_r, M, K)$, $g_r \in L(N)$, $0 \le r < R$. Denote the zero set of $G_r = Z_0(K)g_r$ by ζ_r and set $\zeta = \cap_{r=0}^{R-1}\zeta_r$. Arguing as in the preceding section $f \in L(N)$ is in the linear span of $\mathbf{g}$ if and only if $F = Z_0(K)f$ can be written the form

$$F = \sum_{r=0}^{R-1} F_r, \quad F_r = G_r P_r, \quad P_r \in L(M,K). \tag{21}$$

In fact, if

$$f = \sum_{r=0}^{R-1}\sum_{m'=0}^{K-1}\sum_{n'=0}^{M-1} c^r(m'M, n'K)(g_r)_{m'M,n'K},$$

then we can take

$$P_r = \sum_{m'=0}^{K-1}\sum_{n'=0}^{M-1} c^r(m'M, n'K)e^{-2\pi i(\frac{an'}{M}+\frac{bm'}{K})}.$$

As a consequence, if f is in the linear span of $\mathbf{g}$ then F vanishes on ζ.

Conversely, suppose F vanishes on ζ. The following construction defines the simplest decomposition of F of the form (21). Define $\psi_r \in L(M,K)$, $0 \le r < R$ by

$$\psi_r(a,b) = \begin{cases} 1, & (a,b) \in \zeta_r \\ 0, & (a,b) \notin \zeta_r. \end{cases}$$

Setting

$$\begin{aligned} F_0 &= (1-\psi_0)F \\ F_1 &= \psi_0(1-\psi_1)F \\ &\cdots \\ &\cdots \\ &\cdots \\ F_{R-1} &= \psi_0\psi_1\cdots\psi_{R-2}F \end{aligned}$$

we have

$$F = F_0 + \psi_0 F = F_0 + F_1 + \psi_0\psi_1 F = F_0 + F_1 + \cdots + F_{R-1},$$

where F_r vanishes on ζ_r. Since ζ_r is in the zero set of G_r, we can write $F_r = G_r P_r$, $P_r \in L(M,K)$ and f is in the linear span of $\mathbf{g}$, proving the next result.

Theorem 8 *If* $\mathbf{g}$ *is the disjoint union of critically sampled W-H systems* $\mathbf{g}_r = (g_r, M, K)$, $0 \le r < R$ *and* ζ_r *is the zero set of* $G_r = Z_0(K)g_r$, $0 \le r < R$, *then the dimension of the linear span of* $\mathbf{g}$ *is* $N - J$ *where* J *is the order of* $\zeta = \cap_{r=0}^{R-1}\zeta_r$. *A function* $f \in L(N)$ *is in the linear span of* $\mathbf{g}$ *if and only if* $F = Z_0(K)f$ *vanishes on* ζ.

If we set

$$[G_0(a,b), \cdots, G_{R-1}(a,b)],$$

then we can write

$$F(a,b) = \mathbf{G}(a,b)\begin{bmatrix} P_0(a,b) \\ \cdot \\ \cdot \\ \cdot \\ P_{R-1}(a,b) \end{bmatrix}.$$

Choose $f \in L(N)$ in the linear span of $\mathbf{g}$. An algorithm for computing a W-H expansion of f over $\mathbf{g}$ is given as follows.

- Decompose $F = Z_0(K)f$

$$F = \sum_{r=0}^{R-1} F_r, \quad F_r \in L(M,K)$$

  where F_r vanishes on the zero set ζ_r of G_r, $0 \le r < R$.

- Compute the collection of 2D $M \times K$ FT of

$$P_r(a,b) = \frac{F_r(a,b)}{G_r(a,b)}, \quad 0 \le r < R.$$

This stage is understood to be taken as in the critically sampled case with arbitrary values assigned to the quotient at points where the functions G_r, $0 \leq r < R$ vanish.

If we assume that $TlogT$ computations are needed for the T-point FT then the complexity of one W-H expansion computation is

$$NlogK + R(NlogK + NlogM) + RN \tag{22}$$

but advantage can be taken of the large number of zero data values.

The coefficient set of W-H expansions of $f \in L(N)$ over $\mathbf{g}$ is parameterized by the collection of decompositions of F and by the arbitrarily assigned values to the quotients at the points ζ_r, $0 \leq r < R$.

D.5 Rationally Oversampled

Denote the least common multiple of M and M' by $\overline{M}$ and set $\overline{M} = MS = M'S'$. Then S divides K and $N = \overline{M}\frac{K}{S} = \overline{M}\frac{K'}{S'}$.

Theorem 9 *The rationally oversampled W-H system* $\mathbf{g} = (g, M'K)$ *is the disjoint union of the undersampled W-H systems*

$$\mathbf{g}_{s'} = (g_{s'}, \overline{M}, K), \quad g_{s'} = g_{s'M',0}, \quad 0 \leq s' < S'.$$

Proof We can write $0 \leq m' < K'$ uniquely in the form

$$m' = s' + \overline{m}S', \quad 0 \leq s' < S', \quad 0 \leq \overline{m} < \frac{K'}{S'}.$$

The theorem follows from

$$g_{m'M',n'K} = (g_{s'M',0})_{\overline{m}\overline{M},n'K}, \quad 0 \leq \overline{m} < \frac{K}{S}, \quad 0 \leq n' < M.$$

Consider the undersampled W-H system $(g, \overline{M}, K)$ and set $G = Z_0(K)g$. Since

$$Z(K)(g_{\overline{m}\overline{M},n'K})(a,b) = G(a,b)e^{-2\pi i(\frac{n'}{M}a + \overline{m}\frac{S}{K}b)}, \quad 0 \leq a < M, \quad 0 \leq b < K, \tag{23}$$

$f \in L(N)$ has a W-H expansion over $(g, \overline{M}, K)$ if and only if $F = Z_0(K)f$ can be written as $F = GP$ where $P \in L(M, K)$ satisfies

$$P(a, b + \frac{K}{S}) = P(a,b), \quad 0 \leq a < M, \quad 0 \leq b < K - \frac{K}{S}.$$

For the rationally oversampled W-H system, $\mathbf{g} = (g, M', K)$, set $G_{s'} = Z_0(K)g_{s'}$, where $g_{s'} = g_{s'M',0}$. By theorem 9, and the preceding discussion we have the following result.

Theorem 10 *A function $f \in L(N)$ is in the linear span of* $\mathbf{g}$ *if and only if $F = Z_o(K)f$ has the form*

$$F = \sum_{s'=0}^{S'-1} G_{s'} P_{s'}, \tag{24}$$

where $P_{s'} \in L(M, K)$ satisfies

$$P_{s'}(a, b + \frac{K}{S}) = P_{s'}(a, b), \quad 0 \le s' < S', \quad 0 \le a < M, \ \ 0 \le b < K - \frac{K}{S}. \tag{25}$$

A collection of W-H expansion coefficients of f over $\mathbf{g}$ *is given by the collection of 2D $M \times \frac{K}{S}$ FT of*

$$P_{s'}, \quad 0 \le s' < S'.$$

For each $0 \le a < M$, $0 \le b < \frac{K}{S}$, define $\mathbf{F}(a, b) \in \mathbf{C}^S$ by

$$\mathbf{F}(a, b) = \left[F(a, b + s\frac{K}{S}) \right]_{0 \le s < S},$$

and the $S \times S'$ matrix $\mathbf{G}(a, b)$ by

$$\mathbf{G}(a, b) = \left[G_{s'}(a, b + s\frac{K}{S}) \right]_{0 \le s < S, \, 0 \le s' < S'}.$$

By theorem 10, f is in the linear span of $\mathbf{g}$ if and only if for each $0 \le a < M$, $0 \le b < \frac{K}{S}$, there exists $\mathbf{P}(a, b) \in \mathbf{C}^{S'}$ such that

$$\mathbf{F}(a, b) = \mathbf{G}(a, b)\mathbf{P}(a, b), \quad 0 \le a < M, \ \ 0 \le b < \frac{K}{S}.$$

Denote by $r(a, b)$ the *rank* of $\mathbf{G}(a, b)$. The dimension of the linear span of $\mathbf{g}$ is

$$\sum_{0 \le a < M, 0 \le b < \frac{K}{S}} r(a, b). \tag{26}$$

In particular, if

$$r(a, b) = S, \quad 0 \le a < M, 0 \le b < \frac{K}{S} \tag{27}$$

then $\mathbf{g}$ is complete and every $f \in L(N)$ has a W-H expansion over $\mathbf{g}$.

There are several linear algebra techniques and programming packages that can be applied to characterize the linear span of $\mathbf{g}$ and to compute W-H expansion coefficients for $f \in L(N)$ in this linear span. Gauss elimination is perhaps the most well known technique but QR-decompositions or singular value decompositions (SVD) of $\mathbf{G}(a, b)$ are more suited to applications which subject W-H expansion coefficients to least-square constraints. We will briefly review and introduce notation for SVD at this time.

For each $(a,b) \in 0 \le a < M, 0 \le b < \frac{K}{S}$ the singular value decomposition of $\mathbf{G}(a,b)$ has the form

$$\mathbf{G}(a,b) = \mathbf{U}(a,b)\mathbf{\Sigma}(a,b)\mathbf{V}(a,b)$$

where $\mathbf{U}(a,b)$ is a unitary $S \times S$ matrix, $\mathbf{V}(a,b)$ is a unitary $S' \times S'$ matrix and $\mathbf{\Sigma}(a,b)$ is a 'diagonal' $S \times S'$ matrix

$$\mathbf{\Sigma}(a,b) = \left[\begin{array}{cccc|c} \sigma_0(a,b) & & & & \\ & \sigma_1(a,b) & & & \\ & & \ddots & & 0 \\ & & & \sigma_{S-1}(a,b) & \end{array}\right].$$

Denote the s-column of $\mathbf{U}(a,b)$ by $U_s(a,b)$.

Theorem 11 *A function $f \in L(N)$ is in the linear span of $\mathbf{g}$ if and only if for every (a,b), $0 \le a < M, 0 \le b < \frac{K}{S}$, $\mathbf{F}(a,b)$ is in the linear span of*

$$\{\sigma_s(a,b)U_s(a,b) : 0 \le s < S\}.$$

For f in the linear span of $\mathbf{g}$ we can solve for $\mathbf{P}(a,b)$ by introducing the *pseudo-inverse* of $\mathbf{G}(a,b)$

$$\mathbf{G}^+(a,b) = \mathbf{V}^{-1}(a,b)\mathbf{\Sigma}^+(a,b)\mathbf{U}^{-1}(a,b),$$

where $\mathbf{\Sigma}^+(a,b)$ is the $S' \times S$ diagonal matrix

$$\mathbf{\Sigma}^+(a,b) = \left[\begin{array}{ccc} \sigma_0^+(a,b) & & \\ & \ddots & \\ & 0 \quad 0 & \\ & & \sigma_{S-1}^+(a,b) \\ \hline & 0 & \end{array}\right]$$

with

$$\sigma_s^+(a,b) = \begin{cases} \frac{1}{\sigma_s(a,b)}, & \sigma_s(a,b) \neq 0 \\ 0, & \sigma_s(a,b) = 0, \end{cases} \qquad 0 \le a < M, 0 \le b < \frac{K}{S}.$$

Then

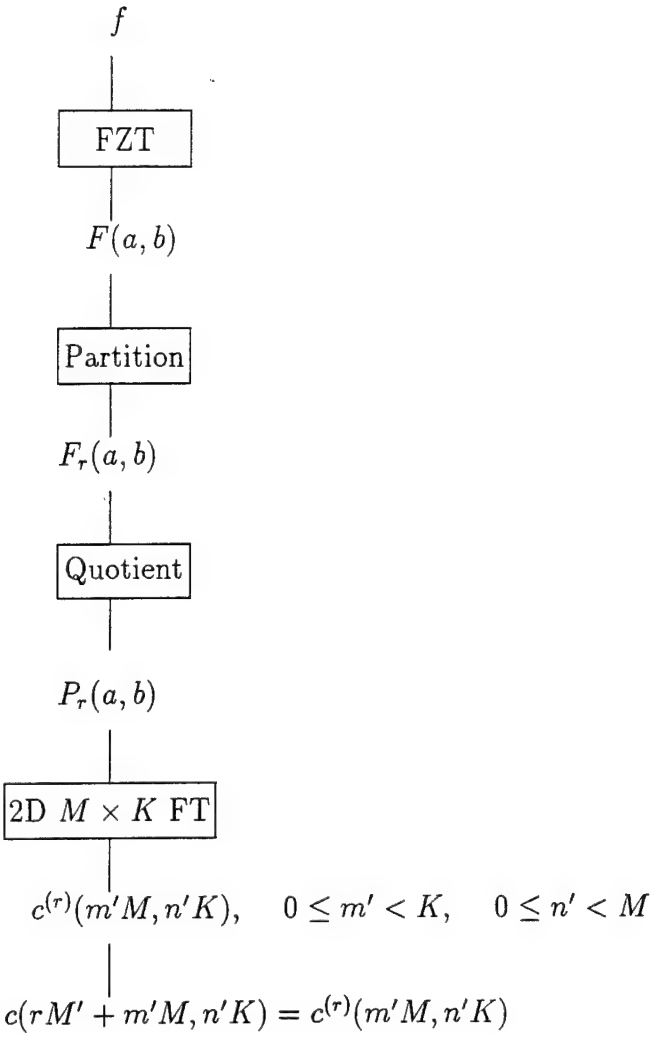
$$\mathbf{P}(a,b) = \mathbf{G}^+(a,b)\mathbf{F}(a,b), \quad 0 \le a < M, 0 \le b < \frac{K}{S}.$$

The multiplicative complexity of the computation is

$$N(logK + S + 1) + \frac{NS'}{S}(log\frac{K}{S} + logM + S')$$

where $NlogN$ is the complexity of the N-point FT and S^2 is the complexity of the action of an $S \times S$ matrix on a vector.

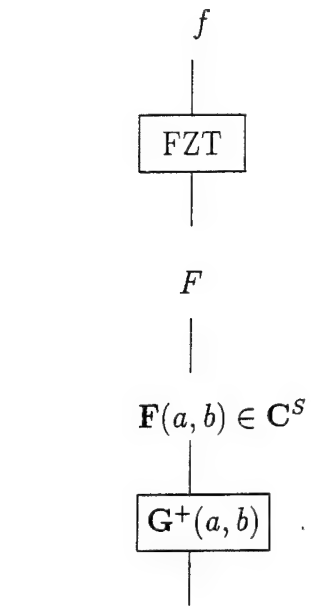
Integer Oversampling: $\mathbf{g} = (g, M', K), \quad M = RM',\ R \in \mathbf{Z}.$



Then

$$f = \sum_{m'=0}^{K'-1} \sum_{n'=0}^{M-1} c(m'M', n'K) g_{m'M', n'K}.$$

Rational Oversampling; $g = (g, M', K)$, $M = RM'$, $R \in Q\, R \notin \mathbf{Z}$.



$$\mathbf{P}(a,b) \in \mathbf{C}^{S'}$$

$$\boxed{\text{2D } M \times \tfrac{K}{S} \text{ FT}}$$

$$P_{s'}(a,b) = \sum_{\overline{m}=0}^{\frac{K}{S}-1} \sum_{n'=0}^{M-1} c_{s'}(\overline{m}, n') e^{-2\pi i(\frac{n'a}{M} + \frac{\overline{m}Sb}{K})}$$

$$c(m', n') = c_{s'}(\overline{m}, n'), \quad m' = s' + \overline{m}S', \quad 0 \le m' < K', \quad 0 \le m' < M.$$

$$f = \sum_{m'=0}^{K'-1} \sum_{n'=0}^{M-1} c(m'n') g_{m'M', n'K}$$

D.6 Implementation Results

In this section we describe implementation issues and present timing results for the implementation of the algorithms presented in the previous sections. Implementations on single Intel i860 RISC microprocessor as well as on the Paragon multi-processor parallel platform are reported.

D.6.1 Critical sampling (C.S.)

We have tested three basic analysis functions:

- Gaussian function

  When K and M are both even integers, the FZT of Gaussian window function has a zero at $(K/2, M/2)$. Set $Q(K/2, M/2) = 0.0$. The total energy of Gabor coefficients will be minimum.

  When either K or M is an odd integer, or both of them are odd integers, the FZT of Gaussian window function has no zeros.

- Rectangular function

  A small size rectangular window will result in FZT with no zeros. For example, $N = K \times M = 1200$, a window of width 90 centered at 600, has no zeros in Zak space.

  A rectangular window of width 150 centered at 600 has zeros in Zak space located at: (j,8), (j,16), (j,24), (j,32), where j=0 to 39.

- Triangular function

  When either K or M is an odd integer, or both of them are odd integers, there are no zeros in Zak space.

  A relatively small triangular window will result in a single zero at the center of Zak space. For example, $N = 40 \times 30 = 1200$, a window of 61 non-zero values centered at 600, has one zero in Zak space at (20, 15).

We have implemented the computation for Critical Sampling case: the main program is in FORTRAN and the FFT modules are fine-tuned i860 assembly with mixed sizes. Timing

results are given in tables 1 and 2.

Complexity

For a real input signal f, the FZT of f is Hermitian symmetric along K dimension. If the analysis signal is also real, then the 2-D $M \times K$ $Q(a,b)$ has the same symmetry. The inverses of the FZT of $g(a,b)$ are pre-computed and stored in memory. The complexity of the computation is ($F(n)$ denotes the complexity of n-point FFT):

$Z(K)f$ (FZT of f)	$M \times$ real $F(K)$
$Z(K)f/Z(K)g$	$K/2 \times M$ multiplications
2-D FT of Q Herm. Symm. along K	$M \times$ Herm. $F(K)$ $K \times$ real $F(M)$

Size N	2-D $K \times M$	Time
256	16×16	0.67
512	16×32	1.20
1024	32×32	2.02
2048	32×64	3.98
4096	64×64	7.41
8192	64×128	14.96
16384	128×128	29.82
32768	128×256	60.89
65536	256×256	125.55
131072	256×512	264.60
262144	512×512	566.99

Table 1: Timing Results (in milliseconds) on the Intel i860 RISC microprocessor (Critical Sampling - 2^k)

D.7 Integer Oversampling

We choose the decomposition $F = Z(K)f = \sum_{r=0}^{R-1} F_r$ such that $F_1, \ldots, F_{R-1}$ each has only one non-zero point, so that the computation of the 2D FT of $Q_1(a,b), \ldots Q_{R-1}(a,b)$ is trivial. The codes are similar to critically sampled case with data rearrangement at the end.

Size N	2-D $K \times M$	Time
384	8×48	1.47
768	16×48	1.99
1536	32×48	3.12
3072	64×48	5.91
3072	128×24	6.15
6144	128×48	12.07
6144	64×96	12.48
12288	512×24	26.07
12288	128×96	24.05
24576	256×96	48.70
49152	256×192	98.71
98304	256×384	203.52
98304	512×192	209.12
196608	512×384	433.41
393216	1024×384	1011.61

Table 2: Timing Results (in milliseconds) on the Intel i860 RISC Microprocessor (Critical Sampling – Mixed sizes)

D.7.1 Rational oversampling

In [9], the authors point out that for Gaussian window function, over-sampled more than 20 percent (5/4), does not have significant influence. We have implemented the computation for oversampling rates 3/2 and 5/4. Again, the main routine is coded in FORTRAN, and the DFT routines are fine-tuned i860 assembly codes for mixed sizes. For the complex singular value decomposition (SVD) we used the LINPACK routine. We have tested three basis functions:

- Gaussian basis function

  Rational oversampling of 3/2 and 5/4 were tested. If the $rank(G(a,b))$ equals to 2 or 4 correspondingly, then $\mathbf{g}$ is complete and every f has a W-H expansion over $\mathbf{g}$.

- Rectangular basis function

  Rational oversampling by 3/2 and 5/4 are tested. Rectangular window sizes have to

be chosen such that it is not a factor of K along K-dimension to have every f expandable in the W-H system.

- Triangular basis function

  An example of size $N = 40 \times 30 = 1200$ has been tested with rational oversampling by 3/2. The experimental results are:

  A window of size 101 centered at 600 results in an expandable W-H system.

  A window of size 151 centered at 600 results in an expandable W-H system.

  A window of size 201 results in point (20,10) being a zero singular value in Zak transform space.

Complexity

In the case of real input and real analysis signals the FZT is Hermitian symmetric along K dimension. We can show that the S' 2-D $M \times K/S$ $P_s(a,b)$ has Hermitian symmetry along K/S dimension. The complexity of real-time computation is:

FZT of f	M x real $F(K)$
$G^+(a,b)F(a,b)$	$M \times K/S$ matrix $S' \times S$ multiply a vector S
S' 2-D FT of P_s with Hermitian Symmetry along K/S	$S' \times M$ x Hermitian $F(K/S)$, $S' \times K/S$ x real $F(M)$

Timing results of various sizes are given in the following tables.

D.8 Parallel Implementation

Assume that a distributed memory parallel computer has p ($< \min(K, M)$) processors. Set

$$P = K/K_1 = M/K_2 \tag{28}$$

Size N	2-D $K \times M$	Time
384	16×24	2.06
768	32×24	2.97
1536	64×24	5.31
3072	64×48	10.79
3072	128×24	10.05
6144	128×48	20.85
6144	64×96	22.86
12288	128×96	43.15
24576	256×96	84.71
49152	256×192	171.39
98304	256×384	412.12
98304	512×192	413.50
196608	512×384	840.02

Table 3: Timing Results (in milliseconds) In the Intel i860 RISC microprocessor (Rational Oversampling (3/2))

The algorithms described in sections 3, 4 and 5 possess highly parallel structure. They are particularly suitable in a distributed memory multiprocessor system. For example, in the critically sampled case, the algorithm can be implemented as follows:

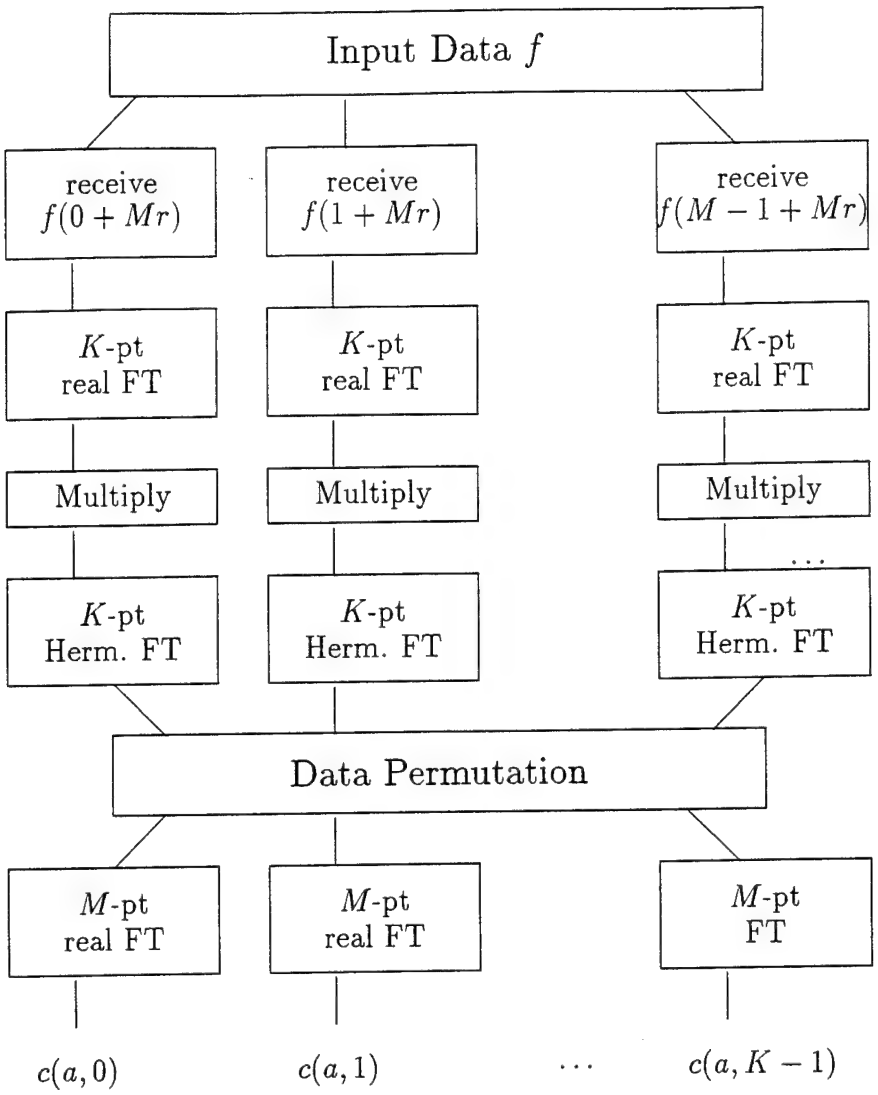
- Each processor receives K_1 K-point input data
- Compute K_1 K-point real FFT
- Point-wise multiplication of the pre-calculated Zak transform of the basis function $1/Z(K)g(a,b)$
- Compute K_1 K-point Hermitian FFT
- Data permutation between processors (matrix transpose)
- Compute K_2 M-point real FFT

Implementation of integer over-sampled case has similar structure as the critically sampled case, and the rationally over-sampled case has a better parallel structure, since it has S' relatively small 2-dimensional $K/S \times M$ FFT's, and they might be carried out locally in each processor without interprocessor data permutation. Timing results of critical sampling on the

Size N	2-D $K \times M$	Time
320	8×40	2.82
640	16×40	3.85
1280	32×40	5.66
2560	64×40	9.65
5120	128×40	16.42
5120	64×80	18.32
10240	128×80	32.09
10240	64×160	37.99
20480	128×160	67.65
40960	128×320	134.08
81920	256×320	258.40
163840	512×320	522.19
327680	512×640	1149.76

Table 4: Timing Results (in milliseconds) on the Intel i860 microprocessor (Rational Oversampling (5/4))

Intel 4-nodes and 8-nodes Paragon are given in tables 6 and 7. The parallel flow diagram is given in Fig. 3.



$$0 \le r < K-1, \quad 0 \le a < M-1$$

Fig. 3. Parallel implementation flow diagram

D.9 Conclusions

Algorithms for the computation of Weyl-Heisenberg (W-H) coefficients for the cases of critical sampling, integer oversampling and rational oversampling have been presented and easily computable conditions for the existence of W-H expansions have been derived in terms of the Zak transform of the signal and the analysis function. We have shown that the algorithms described lead to very efficient FFT based implementations both for single DSP processor systems as well as for parallel multi-processor configurations.

Size N	2-D $K \times M$	Time
16384	128×128	10.06
32768	128×256	19.66
65536	256×256	39.31
131072	256×512	80.24
262144	512×512	163.10
524288	512×1024	368.99
1048576	1024×1024	801.82
2097152	1024×2048	1661.96

Table 5: Timing Results (in milliseconds) on the Intel Paragon (4-nodes)

Size N	2-D $K \times M$	Time
65536	256×256	22.18
131072	256×512	42.45
262144	512×512	86.32
524288	512×1024	189.54
1048576	1024×1024	404.32
2097152	1024×2048	840.17
8388608	2048×2048	1716.03

Table 6: Timing Results (in milliseconds) on the Intel Paragon (8-nodes)

References

[1] L. Auslander and R. Tolimieri, *On Finite Gabor expansion of signals*, IMA Proceedings on Signal Processing, Minneapolis, 1988.

[2] L. Auslander, I. Gertner and R. Tolimieri, *Finite Zak transforms and the finite Fourier transforms*, IMA on Radar and Sonar, Part II, 39 21-36, Springer-Verlag, New York, 1991.

[3] L. Auslander, I. Gertner and R. Tolimieri, *The discrete Zak transform application to time-frequency analysis and synthesis of nonstationary signals*, IEEE Trans. Signal Processing, 39 4 825-835, April 1991.

[4] M. J. Bastiaans, *Gabor signal expansion and degree of freedom of a signal*, Optica Acta, 29 1223-1229, 1982.

[5] D. Gabor, *Theory of communications*, J. IEE, 93, III, 429-457, November 1946.

[6] F. Hlawatsch, *Interference terms in the Wigner distribution*, Proc. 1984 Internat. Conf. on DSP, Florence, Italy, 363-367.

[7] S. Qian and J.M. Morris, *Wigner distribution decomposition and cross-term deleted representation*, Signal Processing, 27, 125-144, Elsevier Publishers, 1992.

[8] S. Raz, *Synthesis of signals from Wigner distributions: Representation on biorthogonal basis*, Signal Processing, 20 4, 303-314, Elsevier Publishers, 1990.

[9] J. Wexler and S. Raz, *Discrete Gabor expansions*, Signal Processing, 21 3, 207-220, Elsevier Publishers, 1990.

[10] J. Wexler and S. Raz, "Wigner-space synthesis of discrete-time periodic signals," *IEEE Trans. on Signal Processing*, **40** 8, August 1990.

[11] M. Zibulski and Y.Y. Zeevi, "Oversampling in the Gabor Scheme," *IEEE Trans. on Signal Processing*, **41** 8, August 1993.

E Group Invariant Fourier Transform Algorithms

E.1 Introduction

The design of algorithms for computing the crystallographic Fourier transform is a subject in applied group theory. In previous works [2, 19] we exploited several elementary results in finite abelian group theory and developed the basic abstract constructs underlying the class of divide and conquer algorithms for computing the multidimensional (MD) discrete Fourier transform (DFT). This setting provides a convenient landscape for introducing a class of divide and conquer crystallographic algorithms. In [2] we outlined a systematic approach for classifying 3-dimensional (3D) crystallographic groups. Applications to 3D crystallography require a detailed understanding of this classification. Similar classifications exist to some extent in higher dimensions and are equally important for applications to quasicrystallography.

The theory developed in this work will operate within the abstract formulation presented in [2, 19]. Finite abelian groups will serve as data indexing sets. A class of *affine group* fast Fourier transform (FFT) algorithms will be introduced which fully utilize data invariance with respect to subgroups of the affine group of data indexing sets. The affine subgroup need not come from a crystallographic group. This approach removes dimension, transform size and crystallographic group from algorithm design and serves to bring out fundamental algorithmic procedures rather than produce an explicit algorithm. These procedures provide tools for writing code which scales over dimension, transform size and crystallographic group and which can be targeted to various architectures. In fact these methods apply to all 230 3D crystallographic groups and to composite transform sizes. We will show the power of these tools by way of an extensive list of implementation examples.

We distinguish three algorithmic strategies. The first is based on the well-known Good-Thomas (GT) or prime factor algorithm which breaks up a FT computation into a sequence of smaller size DFT computations determined by the relatively prime factors of the initial transform sizes. In [2] we developed an abstract formulation of the GT and applied it as a tool for crystallographic algorithms. Our treatment here will be brief and mostly contained in examples.

Reduced transform (RT) algorithms were considered in detail in [2, 19]. A simple generalization of the RT approach based on collections of subgroups will be presented, which provides a universal framework for affine group Fourier transform (FT) algorithms. In applications to 3D crystallography this class of algorithms replaces the problem of computing the FT of 3D group invariant data by that of computing in parallel the FT of collection of 1D or 2D group invariant

data sets. The latter problem is substantially simpler and several efficient implementations are widely practiced. (See appendix).

A third approach, based on a generalization of Cooley-Tukey fast FT (CT FFT), will be discussed which performs generalized periodizations [19] with respect to affine subgroups. This method applies to abelian affine subgroup invariant data and hence to about 100 of the 230 3D crystallographic groups. A CT FFT algorithms associated to an abelian subgroup X of the affine group provides code for Y invariant data with respect to every subgroup Y of X. In applications, we choose X such that the associated CT FFT is easy to code and efficient and such that X contains a large collection of subgroups Y of interest. X itself need not be a crystallographic group. An example will be provided which shows how one code applies to 71 of the crystallographic groups.

This work is organized as follows. In chapter II, we will review all the necessary group theory. Finite abelian group theory will be briefly considered as it is covered in many elementary texts. We reference [19] as it contains all the necessary results. The affine group of a finite abelian group will be defined. Constructs related to the action of affine subgroups on data indexing sets will be introduced. In chapter III we define the Fourier transform of an abelian group and study its fundamental role in interchanging periodization and decimation operations (duality). The RT, CT FFT and GT algorithms are presented in chapter IV as applications of this duality to different global decomposition strategies.

Affine group FFT algorithms based on the RT algorithm are discussed in chapter VI, while those coming from the application of the affine group CT FFT are introduced in chapter VIII. In chapter IX, we briefly sketch a method of incorporating 1D symmetry into FFT computations, which calls on lower order existing FFT routines using the symmetry condition.

Throughout this work, we will provide many examples. These examples have been chosen to reflect both the theory and our experience and others over several years in writing code for the 3D crystallographic FT.

E.2 Group Theory

E.2.1 Finite abelian group

Denote by $\mathbf{Z}/N$ the group of integers modulo N consisting of the set

$$\{0, 1, \cdots, N-1\}$$

with addition taken modulo N. $\mathbf{Z}/N$ is a cyclic group of order N and every cyclic group of order N is isomorphic to $\mathbf{Z}/N$. For example, the multiplicative group U_N of complex Nth roots

of unity

$$\{1, w, \cdots, w^{N-1}\}, \quad w = e^{\frac{2\pi i}{N}},$$

is a cyclic group of order N and the mapping

$$\omega : \mathbf{Z}/N \to U_N$$

defined by $\omega(n) = w^n$, $0 \leq n < N$, is a group isomorphism from $\mathbf{Z}/N$ onto U_N.

The *direct product* of two finite abelian groups

$$A_1 \times A_2$$

is the set of all pairs (a_1, a_2), $a_1 \in A_1$, $a_2 \in A_2$ with componentwise addition. By the *fundamental theorem of finite abelian groups*, every finite abelian group A is isomorphic to a direct product of cyclic groups,

$$A \simeq \mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R. \tag{1}$$

We call Eq. (1) a *presentation* of A. A finite abelian group can have several presentations which vary as to the number of cyclic group factors as well as the orders of the cyclic groups. For example

$$\begin{aligned} \mathbf{Z}/30 &\simeq \mathbf{Z}/2 \times \mathbf{Z}/15 \simeq \mathbf{Z}/3 \times \mathbf{Z}/10 \\ &\simeq \mathbf{Z}/5 \times \mathbf{Z}/6 \simeq \mathbf{Z}/2 \times \mathbf{Z}/3 \times \mathbf{Z}/5 \end{aligned}$$

In general, we have

Theorem E.1 *The direct product of cyclic groups having relatively prime orders is a cyclic group.*

Theorem E.1 is a special case of the Chinese remainder theorem (CRT).

Theorem E.2 *Chinese Remainder Theorem*

Let $N = N_1 N_2 \cdots N_R$ be a factorization of N into pairwise relatively prime integers. Then there exist uniquely determined integers

$$0 \leq e_1, e_2, \cdots, e_R < N$$

satisfying

$$e_r \equiv 1 \, mod \, N_r$$

$$e_r \equiv 0 \, mod \, N_s, \quad 1 \leq r, s \leq R, \quad r \neq s.$$

The set $\{e_1, e_2, \cdots, e_R\}$ is called the *complete system of idempotents* for the factorization $N = N_1 N_2 \cdots N_R$.

Let $\{e_1, e_2, \cdots, e_R\}$ be the complete system of idempotents for the factorization $N = N_1 N_2 \cdots N_R$. By CRT

$$e_r^2 \equiv e_r \, mod \, N, \tag{2}$$

$$e_r e_s \equiv 0 \, mod \, N, \quad 1 \leq r, s \leq R, \;\; r \neq s \tag{3}$$

$$\sum_{r=1}^{R} e_r \equiv 1 \, mod \, N. \tag{4}$$

It follows that every $n \in \mathbf{Z}/N$ has a unique expansion of the form

$$n \equiv n_1 e_1 + n_2 e_2 + \cdots + n_R e_R \;\; mod \;\; N, \quad n_r \in \mathbf{Z}/N_r.$$

In fact

$$n_r \equiv n \, mod \, N_r, \quad 1 \leq r \leq R.$$

CRT shows that the mapping

$$\chi : \mathbf{Z}/N \to \mathbf{Z}/N_1 \times \mathbf{Z}/N_2 \times \cdots \times \mathbf{Z}/N_r.$$

defined by

$$\chi(n) = (n_1, n_2, \cdots, n_r), \quad n_r \equiv n \, mod \, N_r, \;\; 1 \leq r \leq R \tag{5}$$

is an isomorphism having inverse

$$\chi^{-1}(n_1, n_2, \cdots, n_r) = n_1 e_1 + n_2 e_2 + \cdots n_R e_R \, mod \, N. \tag{6}$$

CRT is the basis for many theoretic and applied results in algorithm design. It is a major tool for interchanging between 1D and MD arrays which is the core of the GT algorithm. The use of idempotents in describing this interchange is most important in implementation [19].

CRT can be used to derive the primary factorization of a finite abelian group. Suppose A is a finite abelian group of order N, and we write

$$N = P_1^{\alpha_1} P_2^{\alpha_2} \cdots P_M^{\alpha_M}, \quad \alpha_m \geq 1, \tag{7}$$

where P_1, P_2, $\cdots$, P_M are distinct primes. Choose any presentation of A

$$A \simeq \mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R, \quad N = N_1 \cdots N_R \tag{8}$$

and write

$$N_r = P_1^{\alpha_1(r)} \cdots P_M^{\alpha_M(r)}, \quad \alpha_m(r) \geq 0, \;\; 1 \leq m \leq M.$$

Then

$$\mathbf{Z}/N_r = \mathbf{Z}/P_1^{\alpha_1(r)} \times \cdots \times \mathbf{Z}/P_M^{\alpha_M(r)}$$

and we have, by rearranging factors, the *primary factorization* of A

$$A \simeq A_1 \times \cdots \times A_M, \tag{9}$$

where

$$A_m \simeq \mathbf{Z}/P_m^{\alpha_m(1)} \times \cdots \times \mathbf{Z}/P_m^{\alpha_m(R)}.$$

The primary factorization of A is unique as the factors A_m can be described as the set of all elements in A having order a power of the prime P_m.

E.2.2 Character group

Consider a finite abelian group A of order N. The *character group* A^* of A is the set of all group homomorphisms

$$a^* : A \to U_N$$

with group addition defined by

$$(a^* + b^*)(a) = a^*(a)b^*(a), \quad a^*, b^* \in A^*, a \in A. \tag{10}$$

The character group A^* is the natural indexing set for FT as we can view A as the time parameter space and A^* as the frequency parameter space.

We will usually write $a^*(a)$ as $< a, a^* >$.

The mapping

$$\phi : \mathbf{Z}/N \to (\mathbf{Z}/N)^*$$

defined by

$$< m, \phi(n) >= e^{2\pi i \frac{mn}{N}}, \quad 0 \le n, m < N$$

establishes an isomorphism

$$\mathbf{Z}/N \simeq (\mathbf{Z}/N)^*.$$

More generally, the mapping

$$\phi : \mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R \to (\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R)^*$$

defined by

$$< (m_1, \cdots, m_R), \phi(n_1, \cdots, n_R) >= e^{2\pi i \frac{m_1 n_1}{N_1}} \cdots e^{2\pi i \frac{m_R n_R}{N_R}} \tag{11}$$

establishes an isomorphism

$$\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R \simeq (\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R)^*.$$

By the fundamental theorem, every finite abelian group A is isomorphic to its character group A^*.

Duality

Fix an isomorphism ϕ from A onto A^*. The *dual* $B^\perp$ of a subgroup B of A is defined by

$$B^\perp = \{a \in A : < b, \phi(a) > = 1, \text{ for all } b \in B\}. \tag{12}$$

Since ϕ is an isomorphism,

$$\phi(B^\perp) = \{\phi(b^\perp) : b^\perp \in B^\perp\}$$

is the subgroup of all characters of A that act trivially on B.

Consider the quotient group A/B of B-cosets

$$a + B = \{a + b : b \in B\}$$

with abelian group addition

$$(a + B) + (a' + B) = (a + a') + B.$$

The isomorphism ϕ induces isomorphisms

$$\phi_1 : B^\perp \to (A/B)^*, \quad \phi_2 : A/B^\perp \to B^*,$$

by the formulas

$$< a + B, \phi_1(b^\perp) > = < a, \phi(b^\perp) >, \quad a \in A, \ b^\perp \in B^\perp, \tag{13}$$

$$< b, \phi_2(a + B^\perp) > = < b, \phi(a) >, \quad a \in A, \ b \in B. \tag{14}$$

The characterization of $\phi(B^\perp)$ given above implies both induced isomorphisms are well defined, i.e., independent of coset representation.

The induced isomorphisms ϕ_1 and ϕ_2 play fundamental roles in the description of divide and conquer FT algorithms.

The vector space $L(X)$.

Denote the space of all complex valued functions on a finite set X by $L(X)$. $L(X)$ is a vector space over $\mathbf{C}$ with addition and scalar multiplication defined by

$$(f+g)(x) = f(x) + g(x), \quad f, g \in L(X), \quad x \in X,$$

$$(\alpha f)(x) = \alpha(f(x)), \quad \alpha \in \mathbf{C}, \quad f \in L(X), \quad x \in X.$$

Consider a finite abelian group A and a subgroup B of A. For $f \in L(A)$ define

$$Per_B f(a) = \sum_{b \in B} f(a+b) \tag{15}$$

and

$$Dec_B f(a) = \begin{cases} f(a), & a \in B, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

The *periodization* operator Per_B and the *decimation* operator Dec_B are fundamental operators on $L(A)$.

Suppose A has order N. $L(A)$ has dimension N. The *evaluation basis* of $L(A)$ is the collection of functions

$$\{e_a : a \in A\}$$

defined by

$$e_a(b) = \begin{cases} 1, & b = a, \\ 0, & b \neq a, \end{cases} \quad b \in A. \tag{17}$$

We will denote the evaluation basis by A.

The *character basis* of $L(A)$ is the collection A^* of characters of A. Relative to the *inner product* on $L(A)$ defined by

$$(f, g) = \sum_{a \in A} f(a)\overline{g(a)}, \quad f, g \in L(A), \tag{18}$$

where $\overline{g(a)}$ denotes the complex conjugate of $g(a)$, the evaluation basis is an orthonormal basis of $L(A)$. Since for a^*, $b^* \in A^*$,

$$(a^*, b^*) = \begin{cases} N, & a^* = b^* \\ 0, & a^* \neq b^*, \end{cases}$$

the set

$$\frac{1}{\sqrt{N}} A^*$$

is an orthonormal basis of $L(A)$.

Canonical isomorphism

The evaluation basis A and the character basis A^* are canonical in the sense that they depend solely on group structures and not on presentation. Although the groups A and A^* are isomorphic, there is no canonical isomorphism. Duality is defined relative to a particular choice of isomorphism from A onto A^*. By extension, the groups A and A^{**}, the dual of A^*, are also isomorphic, and in fact a canonical isomorphism can be defined. The canonical isomorphism, as we will see in chapter III, defines the FT of A.

For $a \in A$, the mapping $\Theta(a)$ of A^*

$$\Theta(a)(a^*) =< a, a^* >, \quad a^* \in A^*, \tag{19}$$

is a character of A^*. The mapping

$$\Theta : A \to A^{**} \tag{20}$$

is a canonical isomorphism, since it is defined without reference to presentation.

Consider the evaluation basis A of $L(A)$ and the character basis A^{**} of $L(A^*)$. The canonical isomorphism Θ of A onto A^{**} defines a linear isomorphism $L(\Theta)$ from $L(A)$ onto $L(A^*)$.

E.2.3 Point group

Denote the automorphism group of a finite abelian group A by $Aut(A)$. Subgroups of $Aut(A)$ are called *point groups.*

For a point group H and a point $a \in A$, the *isotropy subgroup* H_a of a in H is defined by

$$H_a = \{\alpha \in H : \alpha(a) = a\}. \tag{21}$$

H_a is a subgroup of H. A point $a \in A$ is called a *fixed point* of H if $H = H_a$. The *H-orbit* of a, denoted by $H(a)$, is defined by

$$H(a) = \{\alpha(a) : \alpha \in H\}. \tag{22}$$

The mapping

$$\alpha \to \alpha(a) : H \to A \tag{23}$$

induces a bijection from the space of right cosets αH_a, $\alpha \in H$, onto $H(a)$.

Fix a group isomorphism $\phi : A \to A^*$. For $\alpha \in Aut(A)$, define the *adjoint* $\alpha^+ \in Aut(A)$ by

$$< a, \phi(\alpha^+(c)) > = < \alpha(a), \phi(c) >, \quad a, c \in A. \tag{24}$$

Set $\alpha^{\#} = (\alpha^{+})^{-1}$, and observe that

$$(\alpha\beta)^{\#} = \alpha^{\#}\beta^{\#}, \quad (\alpha^{-1})^{\#} = (\alpha^{\#})^{-1}.$$

For a point group H, define

$$H^{\#} = \{\alpha^{\#} : \alpha \in H\}.$$

The H-orbit $H(B)$ of a subgroup B of A is the collection of subgroups

$$H(B) = \{\alpha(B) : \alpha \in H\}. \tag{25}$$

Under duality

$$H^{\#}(B^{\perp}) = (H(B))^{\perp}. \tag{26}$$

A collection $\mathcal{B}$ of subgroups of A is called *H-invariant* if

$$h(B) \in \mathcal{B}, \quad h \in H, \ B \in \mathcal{B}.$$

If $\mathcal{B}$ is H-invariant, the action of H partitions $\mathcal{B}$ into disjoint H-orbits. Define a complete system of H-orbit representatives in $\mathcal{B}$ as any collection of subgroups in $\mathcal{B}$

$$B_1, \cdots, B_R$$

such that $\mathcal{B}$ is the disjoint union of the collection of H-orbits

$$H(B_1), \cdots, H(B_R).$$

A *covering* of A is a collection of subgroups $\mathcal{B}$ of A such that

$$A = \cup_{B \in \mathcal{B}} B.$$

Set

$$\mathcal{B}^{\perp} = \{B^{\perp} : B \in \mathcal{B}\}.$$

We say that $\mathcal{B}$ is a *dual covering* of A if $\mathcal{B}^{\perp}$ is a covering of A. We can always construct an H-invariant covering $\mathcal{B}$ of A.

E.2.4 Affine group

The *affine group* of A,

$$Aff(A) = A \leftthreetimes Aut(A), \tag{27}$$

is the set of all (a, α), $a \in A$, $\alpha \in Aut(A)$, with group composition

$$(a, \alpha)(a', \alpha') = (a + \alpha(a'), \alpha\alpha'). \tag{28}$$

$Aff(A)$ acts on A by

$$(a, \alpha)(c) = a + \alpha(c), \quad a, c \in A, \ \alpha \in Aut(A). \tag{29}$$

For $x \in Aff(A)$, we write $x = (a_x, \alpha_x)$, $a_x \in A$, $\alpha_x \in Aut(A)$.

We define two actions of $Aff(A)$ on $L(A)$. For $f \in L(A)$ and $x \in Aff(A)$, define

$$xf(a) = f(x(a)), \quad a \in A. \tag{30}$$

$$x^{\#}f(a) = \overline{< a_x, \phi(\alpha_x^{\#} a) >} f(\alpha_x^{\#} a), \quad a \in A. \tag{31}$$

We say that f is *x-invariant* if $xf = f$ and *$x^{\#}$-invariant* if $x^{\#}f = f$.

Choose a subgroup X of $Aff(A)$. An $f \in L(A)$ is *X-invariant* if f is x-invariant for all $x \in X$, and *$X^{\#}$-invariant* if f is $x^{\#}$-invariant for all $x \in X$.

The *point group* $\dot{X}$ of X is defined by

$$\dot{X} = \{\alpha_x : x \in X\}. \tag{32}$$

$\dot{X}$ is a subgroup of $Aut(A)$, but in general is not contained in X.

E.2.5 Examples

Example E.1 $P6_1$

Crystallographic group $P6_1$ [13] is generated by

$$x = (0, 0, M_2, \alpha))$$

acting on $\mathbf{Z}/3N \times \mathbf{Z}/3N \times \mathbf{Z}/6M$ for natural numbers N and M,

$$x(a_1, a_2, a_3) = (a_1 - a_2, a_1, a_3 + M).$$

Throughout the rest of this example, we will set

$$A = \mathbf{Z}/12 \times \mathbf{Z}/12 \times \mathbf{Z}/12.$$

For $(a_1, a_2, a_3) \in A$,

$$x(a_1, a_2, a_3) = (a_1 - a_2, a_1, a_3 + 2),$$

$$x^2(a_1, a_2, a_3) = (-a_2, a_1 - a_2, a_3 + 4),$$

$$x^3(a_1, a_2, a_3) = (-a_1, -a_2, a_3 + 6),$$

$$x^4(a_1, a_2, a_3) = (a_2 - a_1, -a_1, a_4 + 8),$$

$$x^5(a_1, a_2, a_3) = (a_2, a_2 - a_1, a_3 + 10),$$

$$x^6(a_1, a_2, a_3) = (a_1, a_2, a_3).$$

$P6_1$ acting on A decomposes A into distinct $P6_1$-orbits each of order 6.

$P6_1$ is also a crystallographic group denoted by $P6$ [13]. $P6$ is generated by α.

$$\alpha(a_1, a_2, a_3) = (a_1 - a_2, a_1, a_3).$$

$P6$-orbits also decompose A into distinct orbits. A $P6$-orbit may have 1, 2, 3 or 6 elements.

$$P6(0, 0, a_3) = \{(0, 0, a_3)\}, \quad 0 \leq a_3 \leq 11,$$

and $(0, 0, a_3)$ are fixed points of $P6$.

$$P6(4, 8, a_3) = \{(4, 8, a_3), (8, 4, a_3)\}, \quad 0 \leq a_3 \leq 11.$$

The isotropy subgroup of $(4, 8, a_3)$ is generated by α^2.

$$P6(6, 6, a_3) = \{(6, 6, a_3), (0, 6, a_3), (6, 0, a_3)\}, \quad 0 \leq a_3 \leq 11.$$

The isotropy subgroup of $(6, 6, a_3)$ is generated by α^3.

The non-trivial isotropy subgroups, $\{1, \alpha^2, \alpha^4\}$ and $\{1, \alpha^3\}$, where 1 denotes the identity automorphism, are again crystallographic groups denoted by $P3$ and $P2$ [13], respectively.

With respect to ϕ defined in Eq. (11),

$$< \alpha^{-1}(a_1, a_2, a_3), \phi(b_1, b_2, b_3) >< (a_2, a_2 - a_1, a_3), \phi(b_1, b_2, b_3) >$$

$$= e^{\frac{-2\pi i}{12}(a_2 b_1 + (a_2 - a_1)b_2 + a_3 b_3)}$$

$$= e^{\frac{-2\pi i}{12}(-a_1 b_2 + a_2(b_1 + b_2) + a_3 b_3)}$$

$$=< (a_1, a_2, a_3), \phi(-b_2, b_1 + b_2, b_3) >$$

$$=< (a_1, a_2, a_3), \phi(\alpha^{\#}(b_1, b_2, b_3)) >,$$

and $\alpha^{\#}(b_1, b_2, b_3) = (-b_2, b_1 + b_2, b_3)$.

Example E.2 *P6/mmm*

Crystallographic group $P6/mmm$ is isomorphic to the abstract group

$$\mathbf{Z}/6 \triangleleft \mathbf{Z}/2 \times \mathbf{Z}/2.$$

We will describe the group by listing the 3 generators.

$$\begin{aligned}
&\alpha, \\
&\beta(a_1, a_2, a_3) = (a_2, a_1, -a_3), \quad \beta^{\#} = \beta, \\
&\gamma(a_1, a_2, a_3) = (a_1, a_2, -a_3), \quad \gamma^{\#} = \gamma.
\end{aligned}$$

This is a nonabelian group, and we have the following commuting relations;

$$\beta\alpha = \alpha^{-1}\beta, \quad \gamma\alpha = \alpha\gamma, \quad \gamma\beta = \beta\gamma.$$

Set $A = \mathbf{Z}/12 \times \mathbf{Z}/12 \times \mathbf{Z}/12$. We will consider isotropy subgroups of elements.

$$P6/mmm(4,8,6) = \{(4,8,6),\ (8,4,6)\},$$

$$P6/mmm_{(4,8,6)} = \left\{ \begin{array}{cccccc} 1, & \alpha^2, & \alpha^4, & \gamma, & \alpha^2\gamma, & \alpha^4\gamma, \\ \alpha\beta, & \alpha^3\beta, & \alpha^5\beta, & \alpha\beta\gamma, & \alpha^3\beta\gamma, & \alpha^5\beta\gamma \end{array} \right\}.$$

For $a \neq 0, 6$

$$P6/mmm(4,8,a) = \{(4,8,a),\ (8,4,a), (8,4,-a), (4,8,-1)\}.$$

$$P6/mmm_{(4,8,a)} = \left\{ \begin{array}{ccc} 1, & \alpha^2, & \alpha^4, \\ \alpha\beta\gamma, & \alpha^3\beta\gamma, & \alpha^5\beta\gamma \end{array} \right\} = P3m1,$$

where $P3m1$ is a crystallographic point group.

Example E.3 *Pmmm*

Let $A = \mathbf{Z}/2N \times \mathbf{Z}/2M \times \mathbf{Z}/2L$, for natural numbers N, M and L. $Pmmm < Aut(A)$ is generated by $\rho_1(a_1, a_2, a_3) = (-a_1, a_2, a_3)$, $\rho_2(a_1, a_2, a_3) = (a_1, -a_2, a_3)$, $\rho_3(a_1, a_2, a_3) = (a_1, a_2, -a_3)$. Each of the generators is of order 2 and $Pmmm$ has 8 elements. With respect to the isomorphism defined in Eq. (11),

$$\rho_i^{\#} = \rho_i, \quad i = 1, 2, 3.$$

The subgroup

$$B = \{(b_1 N, b_2 M, b_3 L) : b_i = 0, 1, \quad i = 1, 2, 3\} \tag{33}$$

is the group of fixed points of $Pmmm$. Let

$$B_1 = \{(b_1, b_2M, b_3L) : 0 \le b_1 \le 2N-1, \quad b_2 = 0,1, \quad b_3 = 0,1\}.$$

$$Pmmm_{B_1} = \{1, \rho_2, \rho_3, \rho_2\rho_3\}.$$

$$B_2 = \{(b_1N, b_2, b_3L) : 0 \le b_2 \le 2M-1, \quad b_1 = 0,1, \quad b_3 = 0,1\}.$$

$$Pmmm_{B_2} = \{1, \rho_1, \rho_3, \rho_1\rho_3\}.$$

Example E.4 *Fmmm*

Set $A = \mathbf{Z}/2N \times \mathbf{Z}/2M \times \mathbf{Z}/2L$, for natural numbers N, M and L. The crystallographic affine group $Fmmm < Aff(A)$ is

$$B \times Pmmm,$$

where $B < A$ is the fixed subgroup of $Pmmm$ given in Eq. (33). Each of the generators is of order 2 and $Fmmm$ has 64 elements. An element of $Fmmm$ is of the form

$$(b, \rho_1^{r_1}\rho_2^{r_2}\rho_3^{r_3}), \quad b \in B, \quad r_k = 0,1, \; k = 1,2,3.$$

We will denote the elements of $Fmmm$ by an ordered 6-tuple of 1's and 0's by listing the values of b_j and r_k in order, i.e.,

$$(b_1N, b_2M, b_3L, \rho_1^{r_1}\rho_2^{r_2}\rho_3^{r_3}) \leftrightarrow (b_1, b_2, b_3, r_1, r_2, r_3).$$

In this notation, the group composition in $Fmmm$ is given by componentwise addition modulo 2 in each of the 6 components. We will also index the elements of $Fmmm$ from 0 to 63 by the binary expansion of the 6 tuple,

$$(b_1, b_2, b_3, r_1, r_2, r_3) \leftrightarrow t_1 + 2t_2 + 4t_3 + 8r_1 + 16r_2 + 32r_3.$$

In this notation

$$B = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}.$$

There are no fixed points of $Fmmm$.

$$Fmmm_B = Pmmm.$$

$$Fmmm = Pmmm = \{s_0, s_8, s_{16}, s_{24}, s_{32}, s_{40}, s_{48}, s_{56}\}.$$

E.3 FT of a finite abelian group

View A as a basis of $L(A)$ and A^{**} as a basis of $L(A^*)$. In chapter II, we defined the canonical isomorphism

$$\Theta : A \to A^{**}$$

by

$$\Theta(a)(a^*) =< a, a^* >, \quad a \in A, \ \ a^* \in A^*.$$

The *Fourier transform* F_A of A is the unique linear extension

$$F_A : L(A) \to L(A^*) \tag{34}$$

of Θ. It follows that F_A is a linear isomorphism given by

$$F_A f(a^*) = \sum_{a \in A} f(a) \ < a, a^* >, \quad f \in L(A), \ \ a^* \in A^*, \tag{35}$$

with inverse given by

$$f = \frac{1}{N} \sum_{a^* \in A^*} F_A f(-a^*) a^*, \quad f \in L(A), \ \ N = o(A). \tag{36}$$

The coefficients of f over the character basis are given by $\frac{1}{N} F_A f(-a^*)$, $a^* \in A^*$.

For an isomorphism $\phi : A \to A^*$, define the *FT presentation*

$$F_\phi : L(A) \to L(A) \tag{37}$$

by

$$(F_\phi f)(a) = (F_A f)(\phi(a)), \quad f \in L(A), \ \ a \in A. \tag{38}$$

FT presentations associated to different isomorphisms differ by input permutations. The choice of ϕ can be an important parameter in algorithm design especially in crystallographic applications where ϕ can be matched to crystallographic symmetry to simplify coding. Throughout this chapter we fix an isomorphism $\phi : A \to A^*$.

For a subgroup B of A, define the *induced* Fourier transforms

$$F_{\phi_1} : L(A/B) \to L(B^\perp), \tag{39}$$

$$F_{\phi_2} : L(B) \to L(A/B^\perp), \tag{40}$$

by the formulas

$$(F_{\phi_1} f)(b^\perp) = (F_{A/B} f)(\phi_1(b^\perp)), \quad f \in L(A/B), \ \ b^\perp \in B^\perp, \tag{41}$$

$$(F_{\phi_2} f)(a + B^\perp) = (F_B f)(\phi_2(a + B^\perp)), \quad f \in L(B), \ a \in A, \tag{42}$$

where ϕ_1 and ϕ_2 are defined in Eqs. (13) and (14). F_{ϕ_1} and F_{ϕ_2} are linear isomorphisms. We will write $F^B_{\phi_1}$ for F_{ϕ_1} and $F^B_{\phi_2}$ for F_{ϕ_2} when we want to bring out the dependence on the subgroup B.

E.3.1 Periodization-Decimation

Divide and conquer algorithms for computing the action of F_ϕ decompose the computation into a collection of induced FT computations. In this chapter, we will see how the FT interchanges the fundamental operations of periodization and decimation.

For a subgroup B of A and $f \in L(A)$, $Per_B f \in L(A)$ is B-periodic and we can view $Per_B f \in L(A/B)$.

Theorem E.3 *For $f \in L(A)$, $F_\phi(Per_B f)$ vanishes off of $B^\perp$ and*

$$\begin{aligned}(F_\phi f)(b^\perp) &= \frac{1}{o(B)} F_\phi(Per_B f)(b^\perp) \\ &= F_{\phi_1}(Per_B f)(b^\perp), \quad b^\perp \in B^\perp.\end{aligned}$$

Theorem E.3 implies we compute $F_\phi f$ on the subgroup $B^\perp$ by computing the induced FT $F_{\phi_1}(Per_B f)$.

For $f \in L(A)$, we can view $Dec_B f \in L(B)$.

Theorem E.4 *For $f \in L(A)$, $F_\phi(Dec_B f)$ is $B^\perp$-periodic and*

$$\begin{aligned}Per_{B^\perp}(F_\phi f)(a) &= o(B^\perp) F_\phi(Dec_B f)(a) \\ &= o(B^\perp) F_{\phi_2}(Dec_B f)(a + B^\perp), \quad a \in A.\end{aligned}$$

Theorem E.4 computes the periodization of $F_\phi f$ relative to $B^\perp$ by computing the induced FT $F_{\phi_2}(Dec_B f)$.

E.4 FFT Algorithms

E.4.1 Introduction

Algorithms are distinguished by their strategies for decomposing the global computation. Cooley-Tukey fast FT (CT FFT) algorithms partition the computations into FT of periodizations or

decimations relative to the cosets of some subgroup B of A. Recently formulated Reduced transform (RT) algorithms decompose the computation into FT of periodizations or decimations relative to a collection of subgroups *covering* A. Details including implementation stages on RISC and massively parallel multiprocessors can be found in [14] with performance results.

In this chapter, we will briefly outline the structure of the RT, CT FFT and GT algorithms. Detailed derivations of these algorithms can be found in [2, 19].

E.4.2 RT algorithm

RT algorithms decompose the computation of FT into a collection of induced FT taken over the subgroups of a covering or dual covering of the indexing set. One form of the RT algorithm begins with a dual covering $\mathcal{B}$ of A and computes $F_\phi f$ by

- forming the collection of periodizations

$$Per_B f \in L(A/B), \quad B \in \mathcal{B}$$

- computing the collection of induced FT

$$F^B_{\phi_1}(Per_B f), \quad B \in \mathcal{B}.$$

This completes the computation since $F^B_{\phi_1}(Per_B f)$ equals $F_\phi f$ on $B^\perp$ and $\mathcal{B}$ is a dual covering of A.

A dual form RT algorithm begins with a covering $\mathcal{B}$ of A. For each $a \in A$ define the integer valued function μ on A by

$$\mu(a) = \text{the number of subgroups in } \mathcal{B} \text{ containing } a.$$

Define the *weighted decimations* of f by

$$Dec^\mu_B f(a) = \begin{cases} \frac{1}{\mu(a)} f(a), & a \in B, \\ 0, & \text{otherwise.} \end{cases}$$

Since $\mathcal{B}$ covers A

$$f = \sum_{B \in \mathcal{B}} Dec^\mu_B f \tag{43}$$

$$F_\phi f = \sum_{B \in \mathcal{B}} F_\phi Dec^\mu_B f \tag{44}$$

and we can compute $F_\phi f$ by

- Forming the collection of decimations

$$Dec_B^{\mu} f \in L(B), \quad B \in \mathcal{B}.$$

- Computing the collection of induced FT

$$F_{\phi_2}^{B}(Dec_B^{\mu} f), \quad B \in \mathcal{B}.$$

Redundant computation is a necessary part of RT algorithms. An analysis of the advantages and disadvantages of RT algorithms can be found in [19]. Typically these algorithms are targeted to large size MD DFT computations on shared memory multiprocessors but have been implemented on distributed memory multiprocessors with significant speed-up as compared to standard CT FFT implementations. The RT algorithm on some machines can be bottlenecked by the I/O bandwidth required in the initial stage periodizations but offers complete parallelization (subject to the number of processors and granularity) afterwards and can be easily scaled to transform size and machine configuration. This should be compared with standard approaches which interleave communication and computation by global transpositions.

In applications, say, to the M-dimensional FT, the collection $\mathcal{B}$ is usually taken such that duals are a covering set of K-dimensional ($K < M$) planes through the origin. The dimension K is an important design parameter as it affects local granularity and global parallelism.

E.4.3 CT FFT algorithm

Choose a subgroup $B < A$. One form of the CT FFT begins by subjecting data to *generalized periodizations* relative to B. This step can be implemented by a collection of Fourier transform computations. However we choose to express this step as a collection of generalized periodizations to bring out the analogy with the RT algorithm and to clearly distinguish stages requiring full data access from stages acting, in parallel, on localized data.

Choose a subgroup B of A. For $f \in L(A)$ and $b^* \in B^*$, define $f_{b^*} \in L(A)$ by

$$f_{b^*}(a) = \sum_{b \in B} f(a+b) \; < b, b^* >, \quad a \in A. \tag{45}$$

We call f_{b^*} a *generalized periodization* since

$$f_{b^*}(a+b) = \overline{< b, b^* >} \, f_{b^*}(a), \quad a \in A, \;\; b \in B. \tag{46}$$

Theorem E.5 *For $f \in L(A)$,*

$$f = \frac{1}{o(B)} \sum_{b^* \in B^*} f_{b^*}.$$

$$F_\phi f = \frac{1}{o(B)} \sum_{b^* \in B^*} F_\phi f_{b^*}.$$

It follows that we can compute $F_\phi f$ by computing the collection of FT $F_\phi f_{b^*}$, $b^* \in B^*$.

Consider the group isomorphism $\phi_2 : A/B^\perp \to B^*$. Choose a complete system of $B^\perp$-coset representatives

$$z(b^*) \in \phi_2^{-1}(b^*), \quad b^* \in B^*. \tag{47}$$

Theorem E.6 *$F_\phi f_{b^*}$ vanishes off of the $B^\perp$-coset, $z(b^*) + B^\perp$, and*

$$F_\phi f(z(b^*) + b^\perp) = \frac{1}{o(B)} F_\phi f_{b^*}(z(b^*) + b^\perp), \quad b^\perp \in B^\perp.$$

$F_\phi f_{b^*}$ determines $F_\phi f$ on the $B^\perp$-coset $z(b^*) + B^\perp$, $b^* \in B^*$. Since the $B^\perp$-cosets form a disjoint partition of A, the computations

$$F_\phi f_{b^*}, \quad b^* \in B^*$$

can be implemented in parallel and the second sum in theorem E.5 requires no computation. Once the generalized periodizations are computed, the computation can be completed in parallel by induced FT computations which output $F_\phi f$ on $B^\perp$-cosets. This is accomplished by first performing a twiddle factor multiplication of generalized periodizations defined as follows.

For $b^* \in B^*$, define $g_{b^*} \in L(A)$ by

$$g_{b^*}(a) = f_{b^*}(a) \; < a, \phi(z(b^*)) >, \quad a \in A. \tag{48}$$

g_{b^*} is B-periodic and can be viewed as a function in $L(A/B)$.

Theorem E.7

$$F_\phi f_{b^*}(z(b^*) + b^\perp) = o(B) F_{\phi_1} g_{b^*}(b^\perp), \quad b^\perp \in B^\perp.$$

The CT FFT algorithm combines theorems E.6 and E.7 and computes $F_\phi f$ by independent computations of $F_\phi f$ on the disjoint $B^\perp$-cosets $z(b^*) + B$ by the collection of induced smaller size FT computations

$$F_{\phi_1} g_{b^*}(b^\perp), \quad b^\perp \in B^\perp, \; b^* \in B^*. \tag{49}$$

CT FFT Algorithm

$$f \in L(A)$$

$$\Big|$$

$$f_{b^*} \in L(A), \quad b^* \in B^*$$

$$\Big|$$

$$g_{b^*} \in L(A/B), \quad b^* \in B^*$$

$$\Big|$$

$$F_{\phi_1} g_{b^*} \in L(B^\perp), \quad b^* \in B^*$$

$$F_\phi f(z(b^*) + b^\perp) = F_{\phi_1} g_{b^*}(b^\perp)$$

E.4.4 Good-Thomas algorithm

The GT will be derived as a special case of the CT FFT. In [2, 19], a direct proof was given. Choose a subgroup $B < A$. We require that A has a direct product decomposition.

$$A = B \times C$$

where C is a *subgroup* of A. Choose group isomorphisms

$$\phi_B : B \to B^*, \quad \phi_C : C \to C^*.$$

The mapping

$$\phi : A \to A^*$$

defined by

$$< (b', c'), \phi(b, c) > = < b', \phi(b) > < c', \phi(c) >, \quad b, b' \in B, \quad c, c' \in C$$

is a group isomorphism. Relative to ϕ

$$C = B^\perp, \quad B = C^\perp.$$

Since $A/B = B^\perp$ and $A/B^\perp = B$, $\phi_1^B = \phi_{B^\perp}$ and $\phi_2^B = \phi_B$. In particular, in the notation of the previous chapter, we can take

$$z(b^*) = \phi_B^{-1}(b^*), \quad b^* \in B^*,$$

which amounts to taking B as a complete system of $B^\perp$-coset representatives in A. Under these assumptions, the CT FFT takes the form

$$F_\phi f(b+b^\perp) = \frac{1}{o(B)} F_\phi f_{\phi(b)}(b+b^\perp), \quad b \in B, \ b^\perp \in B^\perp. \tag{50}$$

$$F_\phi f(b+b^\perp) = F_{\phi_{B^\perp}} g_{\phi_B(b)}(b^\perp), \quad b \in B, \ b^\perp \in B^\perp. \tag{51}$$

- Compute
$$g_{\phi_B}(b) \in L(B^\perp), \quad b \in B.$$

- Compute
$$F_{\phi_{B^\perp}}(g_{\phi_B(b)}) \in L(B^\perp), \quad b \in B.$$

The second stage is a collection of FT computations over $B^\perp$. We will see that the first stage is a collection of FT computations over B. By definition

$$g_{\phi_B(b)}(b^\perp) = \sum_{b' \in B} f(b'+b^\perp) < b', \phi(b) >$$

which equals

$$F_{\phi_B} f_{b^\perp}(b)$$

where

$$f_{b^\perp}(b) = f(b+b^\perp), \quad b \in B, \ b^\perp \in B^\perp.$$

The precise statement of the stages of the GT can now be given as follows.

GT algorithm

- Form the slices
$$f_{b^\perp} \in L(B), \quad b^\perp \in B^\perp.$$

- Compute the collection of FT over B
$$F_{\phi_B} f_{b^\perp} \in L(B), \quad b^\perp \in B^\perp.$$

- Form the functions
$$g_{\phi_B(b)} \in L(B^\perp), \quad b \in B.$$

  This step requires data transpose (or permutation).

- Compute the collection of FT over $B^\perp$

$$F_{\phi_{B^\perp}} g_{\phi_B(b)} \in L(B^\perp), \quad b \in B.$$

- Set

$$F_\phi f(b + b^\perp) = F_{\phi_{B^\perp}} g_{\phi_B(b)}(b^\perp).$$

This step requires data transpose (or permutation).

E.5 Examples and implementations

For applications to X-ray crystallography, we will take a 3D case to illustrate the theory presented here. In particular, the smallest non-trivial case, $\mathbf{Z}/12 \times \mathbf{Z}/12 \times \mathbf{Z}/12$ is used in many of the examples, while $\mathbf{Z}/3N \times \mathbf{Z}/3N \times \mathbf{Z}/6M$ and $\mathbf{Z}/2N_1 \times \mathbf{Z}/2N_2 \times \mathbf{Z}/2N_3$ are used in the implementation for several natural numbers.

In the all the examples, we will take the fixed isomorphism ϕ given in Eq. (11). To simplify notation, especially in presenting covering subgroups, we will use the following definition and notation.

Let A be a finite abelian group. For $a \in A$ denote by $< a >$, the subgroup of A *generated* by a,

$$< a >= \{a, 2a, 3a, \cdots, (K-1)a\},$$

where K is the smallest positive integer such that $Ka = 0 \in A$. K is called the *order* of a.

E.5.1 RT algorithm

Two forms of RT algorithm will be derived for $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$. Using CRT, we will extend our current example to groups of the form

$$\mathbf{Z}/3 \cdot 2^N \times \mathbf{Z}/3 \cdot 2^N \times \mathbf{Z}/6M$$

for integers N and M.

Example E.5 *RT algorithm I for* $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$

Set $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$. The following 4 subgroups cover A.

$$B_1^\perp =< (0,1) > \times \mathbf{Z}/3, \quad B_2^\perp =< (1,1) > \times \mathbf{Z}/3,$$

$$B_3^\perp =< (2,1) > \times \mathbf{Z}/3, \quad B_4^\perp =< (1,0) > \times \mathbf{Z}/3,$$

$\mu(a_1, a_2, a_3) = 1$ for all $(a_1, a_2, a_3) \in A$, except $\mu(0,0,0) = 4$. With respect to the isomorphism defined in Eq. (11), we have for $b = 0, 1, 2$

$$B_1 =< (1,0,0) >, \quad B_2 =< (1,2,0) >,$$

$$B_3 =< (1,1,0) >, \quad B_4 =< (0,1,0) > .$$

To index the periodizations, we will fix the coset representatives of A/B_r, $1 \leq r \leq 3$ and A/B_4 as follows.

A/B_r : (0,0,0), (0,1,0), (0,2,0),
(0,0,1), (0,1,1),(0,2,1),
(0,0,2), (0,1,2),(0,2,2), r = 1,2,3.

A/B_4 : (0,0,0), (1,0,0), (2,0,0),
(0,0,1), (1,0,1), (2,0,1),
((0,0,2), (1,0,2), (2,0,0)

For c_1, $c_2 = 0, 1, 2$,

$$f_1(0, c_1, c_2) = \sum_{b=0}^{2} f(b, c_1, c_2), \qquad f_2(0, c_1, c_2) = \sum_{b=0}^{2} f(b, 2b + c_1, c_2),$$

$$f_3(0, c_1, c_2) = \sum_{b=0}^{2} f(b, b + c_1, c_2), \qquad f_4(c_1, 0, c_2) = \sum_{b=0}^{2} f(c_1, b, c_2).$$

The collection of induced FT computations is implemented by the 4 independent 2D 3×3 Fourier transforms.

$$F_{\phi_1}^{B_1} f_1(0, a_1, a_2) = \sum_{c_2=0}^{2} \sum_{c_1=0}^{2} f_1(0, c_1, c_2) e^{\frac{-2\pi i}{3}(a_1 c_1 + a_2 c_2)},$$

$$F_{\phi_1}^{B_2} f_2(a_1, a_1, a_2) = \sum_{c_2=0}^{2} \sum_{c_1=0}^{2} f_2(0, c_1, c_2) e^{\frac{-2\pi i}{3}(a_1 c_1 + a_2 c_2)},$$

$$F_{\phi_1}^{B_3} f_3(2a_1, a_1, a_2) = \sum_{c_2=0}^{2} \sum_{c_1=0}^{2} f_3(0, c_1, c_2) e^{\frac{-2\pi i}{3}(a_1 c_1 + a_2 c_2)},$$

$$F_{\phi_1}^{B_4} f_4(a_1, 0, a_2) = \sum_{c_2=0}^{2} \sum_{c_1=0}^{2} f_4(c_1, 0, c_2) e^{\frac{-2\pi i}{3}(a_1 c_1 + a_2 c_2)}.$$

Example E.6 *RT algorithm II for* $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$

We list a collection of 13 covering subgroups along with their dual groups. Each of the covering subgroups is of order 3, while the dual group is a subgroup of order 9. For $a = 0, 1, 2$ and $b_1, b_2 = 0, 1, 2$,

$$
\begin{aligned}
D_1^\perp &= \{(a,0,0)\}, & D_1 &= \{(0,b_1,b_2)\} \\
D_2^\perp &= \{(0,a,0)\}, & D_2 &= \{(b_1,0,b_2)\} \\
D_3^\perp &= \{(a,a,0)\}, & D_3 &= \{(b_1,2b_1,b_2)\} \\
D_4^\perp &= \{(2a,a,0)\}, & D_4 &= \{(b_1,b_1,b_2)\} \\
D_5^\perp &= \{(2a,2a,a)\}, & D_5 &= \{(b_1,b_2,b_1+b_2)\} \\
D_6^\perp &= \{(0,a,a)\}, & D_6 &= \{(b_1,b_2,2b_2)\} \\
D_7^\perp &= \{(0,2a,a)\}, & D_7 &= \{(b_1,b_2,b_2)\} \\
D_8^\perp &= \{(0,0,a)\}, & D_8 &= \{(b_1,b_2,0)\} \\
D_9^\perp &= \{(a,0,a)\}, & D_9 &= \{(b_1,b_2,2b_1)\} \\
D_{10}^\perp &= \{(2a,0,a)\}, & D_{10} &= \{(b_1,b_2,b_1)\} \\
D_{11}^\perp &= \{(a,a,a)\}, & D_{11} &= \{(b_1,b_2,2b_1+2b_2)\} \\
D_{12}^\perp &= \{(2a,a,a)\}, & D_{12} &= \{(b_1,b_2,b_1+2b_2)\} \\
D_{13}^\perp &= \{(2a,a,2a)\} & D_{13} &= \{(b_1,b_2,2b_1+b_2)\}.
\end{aligned}
$$

$\mu(a_1,a_2,a_3) = 1$ for all $(a_1,a_2,a_3) \in A$, except $\mu(0,0,0) = 13$. We will show 2 of the computations explicitly. The rest follows in exactly the same way. To index the periodizations with respect to D_r, set

$$A/D_3 : \{(0,0,0),(1,0,0),(2,0,0)\}, \tag{52}$$

$$A/D_5 : \{(0,0,0),(0,0,1),(0,0,2)\}. \tag{53}$$

Usually, coset representatives are not unique. Note that although the collection in Eq. (52) can be used as A/D_5 as well as A/D_3, Eq. (53) cannot be used for A/D_3. For $a, c = 0, 1, 2$,

$$Per_{D_3} f(c,0,0) = \sum_{b_2=0}^{2} \sum_{b_1=0}^{2} f(b_1+c, 2b_1, b_2),$$

$$Per_{D_5} f(0,0,c) = \sum_{b_2=0}^{2} \sum_{b_1=0}^{2} f(b_1,b_2,b_1+b_2+c).$$

$$F_{\phi_{1,3}} f_3(a,a,0) = \sum_{c=0}^{2} f_3(c,0,0) e^{\frac{-2\pi i}{3}ac},$$

$$F_{\phi_{1,5}} f_5(2a,2a,a) = \sum_{c=0}^{2} f_5(0,0,c) e^{\frac{-2\pi i}{3}ac}.$$

Remaining cases follow in the same way, and the induced FT computations are implemented by 13 independent 3-point FT.

The above two derivations show uniform decomposition of a 3D problem into 2D and 1D problems, respectively. However, the above two cases can be combined to provide various decompositions.

Example E.7 *RT algorithm for* $A = \mathbf{Z}/2^N \times \mathbf{Z}/2^N$

We will list a collection of covering subgroups of A and their dual subgroups of order 2^N by listing their generators.

A is covered by the following $2^N + 2^{N-1}$ subgroups.

Table E.1 *Covering subgroups of* $\mathbf{Z}/2^N \times \mathbf{Z}/2^N$

	subgroup	generator	dual group generator
$0 \le j < 2^N$	M_j	$(j,1)$	$(-1,j)$
$0 \le l \le 2^{N-1}$	M_{2^N+l}	$(1,2l)$	$(-2l,1)$

To organize the periodizations, we will set

$$A/ < (1,j) > \quad : \quad < (0,1) >, \quad 0 \le j < 2^N,$$

$$A/ < (2l,1) > \quad : \quad < (1,0) >, \quad 0 \le l < 2^{N-1}.$$

For $0 \le c < 2^N$,

$$Per_{B_j} f(0,c) = \sum_{b=0}^{2^N-1} f(b, c+bj), \quad 0 \le j < 2^N,$$

$$Per_{B_{2^N+l}} f(c,0) = \sum_{b=0}^{2^N-1} f(c+b, 2bl), \quad 0 \le l < 2^{N-1}.$$

The collection of induced FT is implemented by $2^N + 2^{N-1}$ independent 2^N-point FT computation.

For the dual RT algorithm, we list the values of the function μ on A with respect to the collection of covering subgroups given in table E.1.

Denote by U_0 the multiplicative units of $\mathbf{Z}/2^N$, i.e.,

$$U_0 = \{a \in \mathbf{Z}/2^N : a \equiv 1 \ \ mod \ \ 2.\}.$$

For $1 \le n \le N-1$, Set

$$U_n = \{a \in \mathbf{Z}/2^N : GCD(a, 2^N) = 2^n\}.$$

Then

$$\mathbf{Z}/2^N = \cup_{n=0}^{N-1} U_n.$$

For $a_n \in U_n$, $a_n \neq 0$,

$$\mu(a_n j, a_n) = 2^n, \quad 0 \leq j < 2^N,$$

$$\mu(a_n, 2a_n l) = 2^n, \quad 0 \leq l < 2^{N-1},$$

$$\mu(0,0) = 2^N + 2^{N-1}.$$

Let $\mathcal{B}$ be the collection of covering subgroups of $\mathbf{Z}/2^N \times \mathbf{Z}/2^N$ given in table E.1. For $B \in \mathcal{B}$, compute

$$Dec_B^{\mu} f.$$

To index the induced FT computations, we will fix $A/B^{\perp}$-coset representatives,

$$A/ < (-1, j) > : < (0,1) >, \quad 0 \leq j \leq 2^N - 1,$$

$$A/ < (-2l, 1) > : < (1,0) >, \quad 0 \leq l \leq 2^{N-1} - 1.$$

The collection of induced FT computation is implemented by $2^N + 2^{N-1}$ independent 2^N-point FT. To complete the computation of F_{ϕ}, we use the periodicity

$$F_{\phi_2}^B(Dec_B^{\mu} f)(a + B^{\perp}) = F_{\phi_2}^B(Dec_B^{\mu} f)(a)$$

and the formula

$$F_{\phi} f = \sum_{B \in \mathcal{B}} F_{\phi_2}(Dec_B^{\mu} f).$$

Example E.8 *Hybrid RT/GT algorithm*

Set $A = \mathbf{Z}/3 \cdot 2^N \times \mathbf{Z}/3 \cdot 2^N$ for a natural number N. By the fundamental theorem,

$$A \simeq A_1 \times A_2, \tag{54}$$

where $A_1 = \mathbf{Z}/2^N \times \mathbf{Z}/2^N$ and $A_2 = \mathbf{Z}/3 \times \mathbf{Z}/3$. The subgroup

$$B = \{(a_1 e_1, a_2 e_1) \in A : 0 \leq a_1, a_2 \leq 2\}$$

is isomorphic to A_1, while

$$B^{\perp} = \{(n_1 e_2, n_2 e_2) \in A : 0 \leq n_1, n_2 \leq 2^N - 1\}$$

is isomorhic to A_2, where e_1 and e_2 are the idempotents associated with the isomorphism in Eq. (54). We have

$$A = B \times B^{\perp}.$$

Using GT algorithm, we can compute F_A by computing F_{A_1} followed by F_{A_2}. The induced FT computations F_{A_1} and F_{A_2} are implemented by RT algorithm.

Example E.9 *Covering subgroup computation via CRT*

Covering subgroups and their dual subgroups for A_2 are given in the following table.

Table E.2 *Covering subgroups of* $\mathbf{Z}/3 \times \mathbf{Z}/3$

k	subgroup	generator	dual group generator
0	L_0	(0,1)	(1,0)
1	L_1	(1,1)	(2,1)
2	L_2	(2,1)	(1,1)
3	L_3	(1,0)	(0,1)

$A_1 \times A_2$ is covered by

$$\{A_1 \times L_k^{\perp} \,:\, 0 \le j \le 3\},$$

while dual subgroups are given by

$$(0,0) \times L_k\} \,:\, 0 \le k \le 3\}.$$

We can also decompose A_1 into covering subgroups. To see this, let $N = 2$.

Table E.3 *Covering subgroups of* $A_1 = \mathbf{Z}/4 \times \mathbf{Z}/4$

j	subgroup	generator	dual group generator
0	M_0	(0,1)	(1,0)
1	M_1	(1,1)	(3,1)
2	M_2	(2,1)	(1,2)
3	M_3	(3,1)	(1,1)
4	M_4	(1,0)	(0,1)
5	M_5	(1,2)	(2,1)

The idempotents in this case are $e_1 = 9$, $e_2 = 4$ and the collection

$$B_{j,k}^{\perp} = 9M_j^{\perp} + 4L_k^{\perp}, \quad 0 \le j \le 5, \;\; 0 \le k \le 3,$$

of 24 subgroups covers A. Each subgroup has order 12, given in table E.4 on the next page.

E.5.2 CT FFT algorithm

Example E.10 *CT algorithm for* $\mathbf{Z}/12$

Set $w = e^{\frac{-2\pi i}{12}}$. For $f \in L(\mathbf{Z}/12)$,

$$(F_\phi f)(b) = \sum_{a=0}^{11} f(a) w^{ab} \quad a, b \in A.$$

For $B = \{0, 4, 8\}$, $B^\perp = \{0, 3, 6, 9\}$, relative to ϕ defined in Eq. (11). Generalized periodization of f gives rise to 3 functions

$$f_{0^*}(a) = f(a) + f(a+4) + f(a+8),$$

$$f_{4^*}(a) = f(a) + w^4 f(a+4) + w^8 f(a+8),$$

$$f_{8^*}(a) = f(a) + w^8 f(a+4) + w^4 f(a+8), \quad a \in \mathbf{Z}/12.$$

Table E.4 *Covering subgroups of* $\mathbf{Z}/12 \times \mathbf{Z}/12$

(j,k)	subgroup	generator	dual group generator
(0,0)	$B_{0,0}$	(0,1)	(1,0)
(1,0)	$B_{1,0}$	(9,1)	(7,9)
(2,0)	$B_{2,0}$	(6,1)	(1,6)
(3,0)	$B_{3,0}$	(3,1)	(1,9)
(4,0)	$B_{4,0}$	(9,4)	(4,9)
(5,0)	$B_{5,0}$	(9,10)	(10,9)
(0,1)	$B_{0,1}$	(4,1)	(5,4)
(1,1)	$B_{1,1}$	(1,1)	(11,1)
(2,1)	$B_{2,1}$	(10,1)	(5,10)
(3,1)	$B_{3,1}$	(7,1)	(5,1)
(4,1)	$B_{4,1}$	(1,4)	(8,1)
(5,1)	$B_{5,1}$	(1,10)	(2,1)
(0,2)	$B_{0,2}$	(8,1)	(1,4)
(1,2)	$B_{1,2}$	(5,1)	(7,1)
(2,2)	$B_{2,2}$	(2,1)	(1,10)
(3,2)	$B_{3,2}$	(11,1)	(1,1)
(4,2)	$B_{4,2}$	(5,4)	(4,1)
(5,2)	$B_{5,2}$	(5,10)	(10,1)
(0,3)	$B_{0,3}$	(4,9)	(9,4)
(1,3)	$B_{1,3}$	(1,9)	(3,1)
(2,3)	$B_{1,3}$	(10,9)	(9,10)
(3,3)	$B_{1,3}$	(7,9)	(9,1)
(4,3)	$B_{4,3}$	(1,0)	(0,1)
(5,3)	$B_{5,3}$	(1,6)	(6,1)

By Eq. (46), $f_{b^*}(a)$ needs to be computed only on a set of B-coset representatives, say, {0, 1, 2, 3}. Thus the periodization is usually implemented by 4 independent 3-point Fourier transform of the strided values of f.

Choosing

$$z(0^*) = 0, \quad z(4^*) = 1, \quad z(8^*) = 2,$$

$$g_{0^*}(a) = f_{0^*}(a),$$

$$g_{4^*}(a) = f_{4^*}(a)\langle a, \phi(1)\rangle = f_{4^*}(a)w^a,$$

$$g_{8^*}(a) = f_{8^*}(a)\langle a, \phi(2)\rangle = f_{8^*}(a)w^{2a}, \quad a \in \mathbf{Z}/12.$$

$\langle a, \phi(z(b^*))\rangle$ is the so-called *twiddle factor* .

The quotient group A/B contains 4 elements, B, $1+B$, $2+B$ and $3+B$. Via the homomorphism ϕ_1 and the B-periodicity of g_{b^*}, we have

$$\begin{aligned} F_\phi f(z(b^*) + b^\perp) &= F_{\phi_1} g_b^*(b^\perp) \\ &= \sum_{a=0}^{3} g_{b^*}(a+B)\langle a+B, \phi_1(b^\perp)\rangle \\ &= \sum_{a=0}^{3} g_{b^*}(a)\langle a, \phi_1(b^\perp)\rangle. \end{aligned}$$

Since $b^\perp = 3b$, for some $b \in A$ and $w^3 = e^{\frac{-2\pi i}{4}}$, the computation of F_ϕ is completed by the 3 independent 4-point Fourier transform of g_{b^*}, $b^* \in B^*$.

Example E.11 *Multidimensional CT FFT*

$$A = \mathbf{Z}/2N_1 \times \mathbf{Z}/2N_2 \times \mathbf{Z}/2N_3.$$

$$\begin{aligned} B &= \{(0,0,0),(N_1,0,0),(0,N_2,0),(N_1,N_2,0), \\ &\qquad (0,0,N_3),(N_1,0,N_3),(0,N_2,N_3),(N_1,N_2,N_3)\} \\ &= \{(b_1N_1, b_2N_2, b_3N_3) : b_n = 0 \text{ or } 1,\ n = 1,2,3\}. \end{aligned} \tag{55}$$

Label the elements of B by b_k, $0 \le k \le 7$ in the order given above.

Table E.5 *Values on B of characters of A.*

	b_0^*	b_1^*	b_2^*	b_3^*	b_4^*	b_5^*	b_6^*	b_7^*
b_0	1	1	1	1	1	1	1	1
b_1	1	−1	1	−1	1	−1	1	−1
b_2	1	1	−1	−1	1	1	−1	−1
b_3	1	−1	−1	1	1	−1	−1	1
b_4	1	1	1	1	−1	−1	−1	−1
b_5	1	−1	1	−1	−1	1	−1	1
b_6	1	1	−1	−1	−1	−1	1	1
b_7	1	−1	−1	1	−1	1	1	−1

Note that the matrix of values of the characters in table E.5 is

$$F(2) \otimes F(2) \otimes F(2),$$

where $\otimes$ denotes the matrix tensor product and $F(2)$ denotes the 2-point FT matrix,

$$F(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

By Eq. (46), we need to compute $f_{b_k^*}$ on a set of B-coset representatives, say,

$$C = \{(a_1, a_2, a_3) : 0 \le a_j \le N_j - 1,\ j = 1, 2, 3\}.$$

Order C antilexicographically. Denote by $\mathbf{f}_0$, the vector of values of f on C listed in order by the ordering of C. Similarly, define the vectors $\mathbf{f}_k$, $0 \le k \le 7$ by listing the values in order of C,

$$\mathbf{f}_k = [f(c + b_k)], \quad c \in C.$$

Then the periodization is obtained by the matrix operation,

$$\begin{bmatrix} \mathbf{f}_{b_0^*} \\ \mathbf{f}_{b_1^*} \\ \mathbf{f}_{b_2^*} \\ \mathbf{f}_{b_3^*} \\ \mathbf{f}_{b_4^*} \\ \mathbf{f}_{b_5^*} \\ \mathbf{f}_{b_6^*} \\ \mathbf{f}_{b_7^*} \end{bmatrix} = (F(2) \otimes I_{N_3} \otimes F(2) \otimes I_{N_2} \otimes F(2) \otimes I_{N_1}) \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \mathbf{f}_4 \\ \mathbf{f}_5 \\ \mathbf{f}_6 \\ \mathbf{f}_7 \end{bmatrix},$$

where I_K denotes the $K \times K$ identity matrix.

$$B^{\perp} = \{(2a_1, 2a_2, 2a_3) : 0 \le a_1 \le N_j - 1,\ j = 1, 2, 3\}.$$

With the following choice of $B^{\perp}$-coset representatives,

$$z(b_{0^*}) = (0,0,0), \quad z(b_{1^*}) = (1,0,0), \quad z(b_{2^*}) = (0,1,0), \quad z(b_{3^*}) = (1,1,0),$$
$$z(b_{4^*}) = (0,0,1), \quad z(b_{5^*}) = (1,0,1), \quad z(b_{6^*}) = (0,1,1), \quad z(b_{7^*}) = (1,1,1).$$

$$\begin{bmatrix} \mathrm{g}_{b_0^*} \\ \mathrm{g}_{b_1^*} \\ \mathrm{g}_{b_2^*} \\ \mathrm{g}_{b_3^*} \\ \mathrm{g}_{b_4^*} \\ \mathrm{g}_{b_5^*} \\ \mathrm{g}_{b_6^*} \\ \mathrm{g}_{b_7^*} \end{bmatrix} = [T] \begin{bmatrix} \mathrm{f}_{b_0^*} \\ \mathrm{f}_{b_1^*} \\ \mathrm{f}_{b_2^*} \\ \mathrm{f}_{b_3^*} \\ \mathrm{f}_{b_4^*} \\ \mathrm{f}_{b_5^*} \\ \mathrm{f}_{b_6^*} \\ \mathrm{f}_{b_7^*} \end{bmatrix},$$

where T is the $8N_1N_2N_3 \times 8N_1N_2N_3$ diagonal matrix whose entry at position $a_1a_2a_3 + kN_1N_2N_3$ is

$$< (a_1, a_2, a_3), z(b_k^*) >, \quad 0 \leq k \leq 7.$$

Since

$$A/B \simeq B^{\perp} \simeq \mathbf{Z}/N_1 \times \mathbf{Z}/N_2 \times \mathbf{Z}/N_3,$$

the induced FT is of size $N_1 \times N_2 \times N_3$ applied to the 8 independent functions $g_{b_k^*}$, $0 \leq k \leq 7$.

E.6 Affine Group RT Algorithms

E.6.1 Introduction

A class of affine group RT algorithms will be constructed which act on data $f \in L(A)$ invariant under the action of affine subgroups $X < Aff(A)$. The effect will be two-fold.

- reduction in the number of required induced FT computations.
- the induced FT computations will be on data invariant under a collection of subgroups of X.

For $x \in Aff(A)$, we define two actions on $L(A)$.

$$xf(a) = f(xa), \tag{56}$$

$$x^{\#}f(a) = \overline{< a_x, \phi(\alpha_x^{\#}a) >} f(\alpha_x^{\#}a). \tag{57}$$

The first main result we have is

Theorem E.8

$$F_{\phi}(xf) = x^{\#}F_{\phi}(f).$$

Proof

$$\begin{aligned} F_\phi(xf)(c) &= \sum_{a\in A}(xf)(a) < a, \phi(c) > \\ &= \sum_{a\in A} f(\alpha_x a + a_x) < a, \phi(c) > \\ &= \sum_{a\in A} f(a) < \alpha_x^{-1}(a - a_x), \phi(c) > \\ &= \overline{< \alpha_x^{-1} a_x, \phi(c) >} \sum_{a\in A} f(a) < \alpha_x^{-1} a, \phi(c) > \\ &= x^{\#} F_\phi(c). \end{aligned}$$

Corollary f is x-invariant if and only if $F_\phi f$ is $x^{\#}$-invariant.

RT algorithms provide a general framework for computing the FT of data invariant under affine subgroups. We begin with data invariant under point groups.

E.6.2 Point group RT algorithm

Choose a dual covering $\mathcal{B}$ of A. The RT algorithm computes $F_\phi f$, $f \in L(A)$, by the collection of induced FT computations

$$F^B_{\phi_1} Per_B f, \quad B \in \mathcal{B}.$$

We will now describe how to modify this form of the RT algorithm when f is invariant under the action of a point group $H < Aut(A)$. This invariance will reduce the number of required induced FT computations to a set of induced FT computations on data invariant under subgroups of H.

Suppose f in H-invariant. Choose a dual covering $\mathcal{B}$ invariant under H,

$$h(B) \in \mathcal{B},\ h \in H,\ B \in \mathcal{B}.$$

The collection of dual subgroups $\mathcal{B}^\perp$ is invariant under $H^{\#}$ and we can choose a subset $\mathcal{B}_0 \subset \mathcal{B}$ such that $\mathcal{B}_0^\perp$ is a complete system of $H^{\#}$-orbit representatives in $\mathcal{B}^\perp$. Since f is H invariant, $F_\phi f$ is $H^{\#}$-invariant and it suffices to compute the the following collection of induced FT.

$$F^B_{\phi_1}(Per_B f), \quad B \in \mathcal{B}_0. \tag{58}$$

This has the effect of reducing the number of induced FT required to complete the computation.

The periodized data $Per_B f$, $B \in \mathcal{B}_0$ inherits some of the data redundancy of f. For a subgroup $B < A$, define

$$H_B = \{h \in H : h(B) = B\}.$$

H_B induces a group of automorphisms of A/B by

$$h(a+B) = ha + B, \quad h \in H_B, \ \ a \in A.$$

Theorem E.9 *If f is H-invariant and B is a subgroup of A, then*

$$Per_B f(ha) = Per_{h^{-1}(B)} f(a), \quad a \in A, \ \ h \in H.$$

In particular, $Per_B f \in L(A/B)$ is H_B-invariant.

By the theorem, the induced FT in Eq.(58) is computed on H_B-invariant $Per_B f$, $B \in \mathcal{B}_0$. To make full use of the H-invariance of f we must supply code which makes full use of this H_B. In crystallogrphic applications we can choose $\mathcal{B}$ such that A/B is 1-D or 2-D. Standard point group FFT algorithms can be applied in the 1D case (see appendix). 2D point group invariant FFT algorithms have recently been implemented using variants of Winograd's multiplicative FFT [3, 5].

H-invariant RT algorithm Choose a dual covering $\mathcal{B}$ of A invariant under H and a complete system of H-orbit representatives $\mathcal{B}_0$ in $\mathcal{B}$.

- Form the periodizations
$$Per_B f \in L(A/B), \quad B \in \mathcal{B}_0.$$

- Compute the H_B-invariant induced FT's
$$F^B_{\phi_1}(Per_B f), \quad B \in \mathcal{B}_0.$$

- Compute
$$F^B_{\phi_1}(Per_B f), \quad B \in \mathcal{B},$$
by $H^{\#}$-invariance.

Example E.12 *P6-invariant RT algorithm I*

Set
$$A_3 = \mathbf{Z}/6M, \quad A = \mathbf{Z}/3 \cdot 2^N \times \mathbf{Z}/3 \cdot 2^N \times A_3,$$
for integers N and M. Using the Chinese remainder theorem, we can write A as

$$(e_1 A_1 + e_2 A_2) \times A_3,$$

where $A_1 \simeq \mathbf{Z}/2^N \times \mathbf{Z}/2^N$ and $A_2 \simeq \mathbf{Z}/3 \times \mathbf{Z}/3$. A is covered by the following collection of subgroups, where L_k, $k = 0, 1, 2, 3$ are given in table E.2.

$$B_k^{\perp} = e_1 A_1 + e_2 L_k^{\perp} \times A_3.$$

$$B_k = \{(0,0)\} + e_2 L_k \times \{0\}.$$

$$P6^{\#}(B_0^{\perp}) = \{B_0^{\perp}, B_1^{\perp}, B_3^{\perp}\} \quad P6^{\#}(B_2^{\perp}) = \{B_2^{\perp}\},$$

and $\{B_k^{\perp} : 0 \le k \le 3\}$ is a $P6^{\#}$-invariant covering of A. Hence for $P6$-invariant $f \in L(A)$, we need to compute $F_A f$ only on $B_0^{\perp}$ and $B_2^{\perp}$.

$$f_0 = Per_{B_0}, \quad f_2 = Per_{B_2} f.$$

To index the periodization, set

$$A/B_r : A_1 + e_2 L_1^{\perp}, \quad r = 0, 1,$$

$$A/B_s : A_1 + e_2 L_4^{\perp}, \quad s = 2, 3.$$

For $0 \le n_1, n_2 \le N-1$, $0 \le k \le 2$, $0 \le m \le 6M-1$,

$$f_0(e_1 n_1 + e_2 k, e_1 n_2, m) = \sum_{a=0}^{2} f(e_1 n_1 + e_2 k, e_1 n_2 + e_2 a, m),$$

$$f_2(e_1 n_1, e_1 n_2 + e_2 k, m) = \sum_{a=0}^{2} f(e_1 n_1 + e_2(k+2a), e_1 n_2 + e_2 a, m).$$

$$\begin{aligned} f_0(\alpha^3(e_1 n_1 + e_2 k, e_1 n_2, m)) &= f_1(-e_1 n_1 - e_2 k, -e_1 n_2, m), \\ &= \sum_{a=0}^{2} f(-e_1 n_1 - e_2 k, -e_1 n_2 + e_2 a, m), \\ &= \sum_{a=0}^{2} f(e_1 n_1 + e_2 k, e_1 n_2 - e_2 a, m), \\ &= f_1(e_1 n_1 + e_2 k, e_1 n_2, m), \end{aligned}$$

$$\begin{aligned} f_2(\alpha(e_1 n_1, e_1 n_2 + e_2 k, m)) &= f_3(-e_1 n_2, e_1 n_2 + e_2 k - e_1 n_2, m) \\ &= \sum_{a=0}^{2} f(-e_1 n_2 + 2e_2 a, e_1 n_2 + e_2 k - e_1 n_2 + e_2 a, m) \\ &= \sum_{a=0}^{2} f(e_1 n_1 + e_2 k + 2e_2 a, e_1 n_2 + e_2 a, m) \\ &= f_3(e_1 n_1, e_1 n_2 + e_2 k, m) \end{aligned}$$

$$P6_{B_0} = P6_{B_1} = P6_{B_3} = \{1, \alpha^3\} = P2, \quad P6_{B_2} = P6.$$

The induced FT computations $F_{\phi_1}^{B_0}$ and $F_{\phi_1}^{B_2}$ are made on $P2$ and $P6$ invariant data, respectively.

Example E.13 *P6-invariant RT algorithm II.*

We can further reduce invariance condition on the periodized functions by applying RT on A_1. To this end, we will set $A_1 = \mathbf{Z}/4 \times \mathbf{Z}/4$, and use the covering subgroups that are given in table E.4. The collection

$$\mathcal{B} = D_{j,k}^{\perp} = B_{j,k}^{\perp} \times A_3, \quad 0 \le j \le 5, \ \ 0 \le k \le 3$$

covers

$$\mathbf{Z}/12 \times \mathbf{Z}/12 \times A_3.$$

The dual subgroups are given by

$$D_{j,k} = B_{j,k} \times \{0\}, \quad 0 \le j \le 5, \ \ 0 \le k \le 3.$$

Let $\alpha^{\#} M_j \times A_3 = M_{j'} \times A_3$ and $\alpha^{\#} L_k \times A_3 = L_{k'} \times A_3$. Then we have

$$\alpha^{\#} \left((e_1 M_j + e_2 L_k) \times A_3 \right) = (e_1 M_{j'} + e_2 L_{k'}) \times A_3.$$

Thus to compute the $P6^{\#}$-orbit decomposition of $\mathcal{B}$, we first decompose the collections $\{M_j \times A_3 : 0 \le j \le 5\}$ and $\{L_k \times A_3 : 0 \le k \le 4\}$ independently, then place the decomposition into $\mathcal{B}$ by CRT.

Table E.6 *$P6^{\#}$-orbit decomposition of subgroups in $\mathbf{Z}/4 \times \mathbf{Z}/4$*

$(0,1)$	$\alpha^{\#}(0,1) = (3,1)$	$\alpha^{\#}(3,1) = (3,0)$
$< (0,1) >$	$< (3,1) >$	$< (3,0) > = < (1,0) >$
M_0	M_3	M_4
$(1,1)$	$\alpha^{\#}(1,1) = (3,2)$	$\alpha^{\#}(3,2) = (2,1)$
$< (1,1) >$	$< (3,2) > = < (1,2) >$	$< (2,1) >$
M_1	M_5	M_2

Table E.7 *$P6^{\#}$-orbit decomposition of subgroups in* $\mathbf{Z}/3 \times \mathbf{Z}/3$

$(0,1)$	$\alpha^{\#}(0,1) = (2,1)$	$\alpha^{\#}(2,1) = (2,0)$
$<(0,1)>$	$<(2,1)>$	$<(2,0)>=<(1,0)>$
L_0	L_2	L_3

$(1,1)$	$\alpha^{\#}(1,1) = (2,2)$
$<(1,1)>$	$<(2,2)>=<(1,1)>$
L_1	

We have the following $P6^{\#}$-orbit decomposition of A.

$$P6^{\#}(D^{\perp}_{0,0}) = \{D^{\perp}_{0,0}, D^{\perp}_{3,2}, D^{\perp}_{4,3}\}, \quad P6^{\#}(D^{\perp}_{3,0}) = \{D^{\perp}_{3,0}, D^{\perp}_{4,2}, D^{\perp}_{0,3}\},$$
$$P6^{\#}(D^{\perp}_{4,0}) = \{D^{\perp}_{4,0}, D^{\perp}_{0,2}, D^{\perp}_{3,3}\}, \quad P6^{\#}(D^{\perp}_{1,0}) = \{D^{\perp}_{1,0}, D^{\perp}_{5,2}, D^{\perp}_{2,3}\},$$
$$P6^{\#}(D^{\perp}_{5,0}) = \{D^{\perp}_{5,0}, D^{\perp}_{2,2}, D^{\perp}_{1,3}\}, \quad P6^{\#}(D^{\perp}_{2,0}) = \{D^{\perp}_{2,0}, D^{\perp}_{1,2}, D^{\perp}_{5,3}\},$$
$$P6^{\#}(D^{\perp}_{0,1}) = \{D^{\perp}_{0,1}, D^{\perp}_{3,1}, D^{\perp}_{4,1}\}, \quad P6^{\#}(D^{\perp}_{1,1}) = \{D^{\perp}_{1,1}, D^{\perp}_{5,1}, D^{\perp}_{2,1}\}$$

We will choose as $P6^{\#}$-orbit representatives,

$$\mathcal{B}_0 = \{D^{\perp}_{0,0}, D^{\perp}_{1,0}, D^{\perp}_{2,0}, D^{\perp}_{3,0}, D^{\perp}_{4,0}, D^{\perp}_{5,0}, D^{\perp}_{0,1}, D^{\perp}_{1,1}\}. \tag{59}$$

It is easy to show that the periodizations of $P6$-invariant $f \in L(A)$ with respect to the duals of the above $P6^{\#}$-orbits representatives are $P2$-invariant, and the induced FT computations are made on this invariant data.

Let $\hat{f}$ be the FT of a $P6$-invariant function $f \in L(A)$. $\hat{f}$ on $D_{j,k} \in \mathcal{B}_0$ is determined by the induced FT of $D_{j,k}$-periodized function $f_{D_{j,k}}$. By the $P6^{\#}$-invariance of $\hat{f}$, for example, $\hat{f}$ of $D^{\perp}_{0,0}$ determines $\hat{f}$ on $D^{\perp}_{3,2}$ and $\hat{f}$ on $D^{\perp}_{4,3}$.

$$\hat{f}(0,1,m) = \hat{f}(11,1,m) = \hat{f}(1,0,m),$$

$$(0,1,m) \in D^{\perp}_{0,0}, \quad (11,1,m) \in D^{\perp}_{3,2}, \quad (11,0,m) \in D^{\perp}_{4,3}.$$

Example E.14 *$P3$-invariant RT algorithm*

Crystallographic group $P3$ is generated by α^2. Since $P3$ is a subgroup of $P6$, $P6^{\#}$-invariant covering of

$$\mathbf{Z}/12 \times \mathbf{Z}/12 \times A_3.$$

is also $P3$-invariant. In fact, the $P3^{\#}$-orbits and the $P6^{\#}$-orbits of the covering subgroups are the same. Thus as in the case of $P6$, the induced FT's are computed only on the collection $\mathcal{B}_0$. However, the periodized functions have only the trivial invariance, and symmetry specific FT routines are not required.

Example E.15 *$P6/mmm$-invariant covering for* $\mathbf{Z}/12 \times \mathbf{Z}/12 \times A_3$.

The above two examples lead to the following unifying strategy.

> *Choose a point group H that contains sufficiently many subgroups. Since $H^{\#}$-invariant covering is invariant under any subgroup $K^{\#} < H^{\#}$, for K-invariant data, RT algorithm proceeds by disabling the computations except on the $K^{\#}$-orbit representatives.*

As an example, we will consider the crystallographic $P6/mmm$ which contains all the trigonal and hexagonal point groups, which comprises 16 of the 53 3D crystallographic point groups.

$$\begin{aligned}
P6/mmm^{\#}(D^{\perp}_{1,0}) &= \{D^{\perp}_{1,0}, D^{\perp}_{5,2}, D^{\perp}_{2,3}, D^{\perp}_{5,0}, D^{\perp}_{2,2}, D^{\perp}_{1,3}\},\\
P6/mmm^{\#}(D^{\perp}_{3,0}) &= \{D^{\perp}_{3,0}, D^{\perp}_{4,2}, D^{\perp}_{0,3}, D^{\perp}_{4,0}, D^{\perp}_{0,2}, D^{\perp}_{3,3}\},\\
P6/mmm^{\#}(D^{\perp}_{0,0}) &= \{D^{\perp}_{0,0}, D^{\perp}_{3,2}, D^{\perp}_{4,3}\},\\
P6/mmm^{\#}(D^{\perp}_{2,0}) &= \{D^{\perp}_{2,0}, D^{\perp}_{1,2}, D^{\perp}_{5,3}\},\\
P6/mmm^{\#}(D^{\perp}_{0,1}) &= \{D^{\perp}_{0,1}, D^{\perp}_{3,1}, D^{\perp}_{4,1}\},\\
P6/mmm^{\#}(D^{\perp}_{1,1}) &= \{D^{\perp}_{1,1}, D^{\perp}_{5,1}, D^{\perp}_{2,1}\}.
\end{aligned}$$

A collection $P6/mmm^{\#}$-orbit representatives is

$$\{D^{\perp}_{0,0}, D^{\perp}_{1,0}, D^{\perp}_{2,0}, D^{\perp}_{3,0}, D^{\perp}_{0,1}, D^{\perp}_{1,1}\}$$

and the computation is required only on this collection of subgroups for a $P6/mmm$-invariant functions. To simplify notation, set $H_{j,k} = P6/mmm_{D^{\perp}_{j,k}}$, the invariant group of the $D^{\perp}_{j,k}$-periodized functions.

$$H_{0,0} = H_{2,0} = H_{0,1} = H_{1,1} = \{1, \alpha^3, \beta, \alpha^3\beta, \gamma, \alpha^3\gamma, \beta\gamma, \alpha^3\beta\gamma\}.$$

$$H_{1,0} = H_{3,0} = \{1, \alpha^3, \gamma, \alpha^3\gamma, \}.$$

The induced FT computations are made on the $H_{0,0}$ or $H_{1,0}$-invariant functions.

Example E.16 *Implementation of RT with respect to P6/mmm*

$$A = \mathbf{Z}/3 \cdot 2^N \times \mathbf{Z}/3 \cdot 2^N \times \mathbf{Z}/6M.$$

By the fundamental theorem,

$$A \simeq \mathbf{Z}/2^N \times \mathbf{Z}/2^N \times \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3 \cdot 2^M.$$

Let e_1 and e_2 be the system of idempotents associated with the isomorphism

$$\mathbf{Z}/3 \cdot 2^N \simeq \mathbf{Z}/2^N \times \mathbf{Z}/3$$

and again set $A_3 = \mathbf{Z}/6M$.

$$\mathcal{D} = (e_1 L_k^{\perp} + e_2 M_j^{\perp}) \times A_3,$$

where $L_k^{\perp}$ and $M_j^{\perp}$ are collection of covering subgroups in $\mathbf{Z}/2^N \times \mathbf{Z}/2^N$ and $\mathbf{Z}/3 \times \mathbf{Z}/3$, respectively as listed in tables E.2 and E.1. For easier reference, we repeat the tables here.

Table E.1 *Covering subgroups of* $\mathbf{Z}/2^N \times \mathbf{Z}/2^N$

	subgroup	generator	dual group generator
$0 \le j < 2^N$	M_j	$(j,1)$	$(-1,j)$
$0 \le l \le 2^{N-1}$	M_{2^N+l}	$(1,2l)$	$(-2l,1)$

We will denote this collection by $\mathcal{B}$.

Table E.2 *Covering subgroups of* $\mathbf{Z}/3 \times \mathbf{Z}/3$

k	subgroup	generator	dual group generator
0	L_0	(0,1)	(1,0)
1	L_1	(1,1)	(2,1)
2	L_2	(2,1)	(1,1)
3	L_3	(1,0)	(0,1)

It is straightforward show that $\mathcal{D}$ is a $P6/mmm^{\#}$-invariant dual covering of A. We will give the $P6/mmm^{\#}$-orbit decomposition of $\mathcal{D}$. Recall $\beta^{\#} = \beta$ and $\gamma^{\#} = \gamma$.

$P6/mmm^{\#}$-orbit structure in $\mathbf{Z}/3 \times \mathbf{Z}/3$ is the same as that of $P3^{\#}$, since actions by β or γ does not change the orbit structure.

$$P6/mmm^{\#}(L_0) = \{L_0, L_2, L_3\}, \quad P6/mmm^{\#}(L_1) = \{L_1\}.$$

$$\beta(L_0) = L_3, \quad \beta(L_1) = L_1, \quad \beta(L_2) = L_2.$$

$P6^{\#}$-orbit of $< (j,1) >$,

$$P6^{\#} < (j,1) >= \{< (j,1) >, < (-1,j+1) >, < (-j-1,j) >\}$$

contains three distinct subgroups. To see this, note first

$$< (-1,j+1) >= < (1,-j-1) >,$$

$$< (-j-1,j) >= < (j^{-1}(-j-1),1) > .$$

As j ranges through U_0, $j^{-1}(-j-1)$ ranges through $\mathbf{Z}/2^N - U_0$, and $-j-1$ ranges through $2l$, $0 \leq l \leq 2^{N-1}-1$. In fact, we have the following partitioning of $\mathcal{B}$ into $P6^{\#}$-orbits.

$$\bigcup_{j \in U_0} \{< (j,1) >, < (-1,j+1) >, < (-j-1,j) >\}.$$

β maps $< (j,1) >$ onto $< (1,j) >$. We will show that there are exactly 4 subgroups of the form $< (j,1) >$ with $j \in U_0$ that are β-invariant. Suppose

$$< (j,1) >= < (1,j) >= < (j^{-1},1) > .$$

Then $j^2 \equiv 1 \bmod 2^N$. $j \in U_0$ can be written as $2l+1$, $0 \leq l \leq 2^{N-1}-1$. In terms of l, the following congruences hold.

$$\begin{aligned} (2l+1)^2 = 4l^2 + 4l + 1 &\equiv 1 \quad mod\ 2^N. \\ 4l(l+1) &\equiv 0 \quad mod\ 2^N. \\ l(l+1) &\equiv 0 \quad mod\ 2^{N-2}. \end{aligned}$$

The last congruence has exactly 4 solutions for $0 \leq l \leq 2^{N-1}-1$,

$$\begin{cases} l = 0, & \Longrightarrow \ j = 1, \\ l = 2^{N-2}-1, & \Longrightarrow \ j = 2^{N-1}-1, \\ l = 2^{N-2}, & \Longrightarrow \ j = 2^{N-1}+1, \\ l = 2^{N-1}-1, & \Longrightarrow \ j = 2^N - 1. \end{cases}$$

Partitioning of $\mathcal{B}$ into $P6/mmm^{\#}$-orbits is given below.

- $1 \leq j \leq 2^{N-1}-3$,

  $\{< (2j+1,1) >, < (-1,2j+2) >, < (-2j-2,2j+1) >,$

  $< (1,2j+1) >, < (2j+2,-1) >, < (2j+1,-2j-2) >\}$,

- $\{ < (1,1) >, < (-1,2) >, < (-2,1) > \}$,
  $\{ < (2^{N-1}-1,1) >, < (-1,2^{N-1}) >, < (-2^{N-1},2^{N-1}-1) > \}$,
  $\{ < (2^{N-1}+1,1) >, < (-1,2^{N-1}+2) >, < (-2^{N-1}-2,2^{N-1}+1) > \}$.
  $\{ < (-1,1) >, < (-1,0) >, < (0,-1) > \}$.

There are 2^{N-1} $P6/mmm^{\#}$-orbits in $\mathcal{B}$, 4 of which contain 3 subgroups. Action by γ does not change the orbit structure.

We list two examples of $P6/mmm^{\#}$-orbits in $\mathcal{D}$.

Set $l = 2j+1$. From the orbit of $< (l,1) >$ in $\mathcal{B}$ and L_0, we obtain

$< (e_1 l, 1) > \times A_3$, $< (-e_1 + 2e_2, e_1 l + e_2) > \times A_3$,

$< (-e_1 l + e_2, e_1 l) > \times A_3$, $< (1, e_1 l) > \times A_3$,

$< (e_1 l + 2e_2, -e_1 + e_2) > \times A_3$, $< (e_1 l, -e_1 l + e_2) > \times A_3$.

from the orbit of $< (1,1) >$ and L_0, we obtain

$< (e_1, 1) > \times A_3$, $< (-e_1 + 2e_2, e_1 + 1) > \times A_3$, $< (-2e_1 + e_2, e_1) > \times A_3$.

In $\mathcal{D}$, there are $4 \cdots 2^{N-1}$ $P6/mmm^{\#}$-orbits, 4 of which contain 3 subgroups, the rest contain 6 subgroups.

For completeness, we list the values of idempotents.

(1) If $2^N \equiv 1 \ \ mod \ 3$, then

$$e_1 = 2^{N+1} + 1, \quad e_2 = 2^N.$$

(2) If $2^N \equiv 2 \ \ mod \ 3$, then

$$e_1 = 2^N + 1, \quad e_2 = 2^{N+1}.$$

Choose a $P6/mmm$-invariant function $f \in L(A)$. By the invariance, the induced FT computation only on a collection of $P6/mmm^{\#}$-orbit representatives determines the FT of f. As in example 6.4, the periodized functions are invariant under one of the two subgroups of $P6/mmm$, $H_{0,0}$, or $H_{1,0}$. Specifically, a periodized function f_D is $H_{1,0}$-invariant if the $P6/mmm^{\#}$ orbit of D contains 6 subgroups, while f_D is $H_{0,0}$-invariant if the $P6/mmm^{\#}$ orbit of D contains 3 subgroups.

E.6.3 Affine group RT algorithm

Choose a subgroup X of $Aff(A)$ and denote the point group of X by $\dot{X}$. For X-invariant $f \in L(A)$ we have

$$F_\phi f(\alpha_x^\# a) =< a_x, \phi(\alpha_x^\# a) > F_\phi f(a), \quad a \in A, \ \ x \in X. \tag{60}$$

$F_\phi f$ is not invariant under $\dot{X}^\#$ but $F_\phi f(a)$ determines $F_\phi f$ at each point in the $\dot{X}^\#$-orbit of a.

Choose an $\dot{X}$-invariant dual covering $\mathcal{B}$ of A and a complete system $\mathcal{B}_0$ of $\dot{X}$-orbit representatives in $\mathcal{B}$. $\mathcal{B}_0^\perp$ is a complete system of $\dot{X}^\#$ representatives in the covering $\mathcal{B}^\perp$ of A. In the presence of X-invariance, the RT algorithm can be implemented by first computing the induced FT

$$F_{\phi_1}^B(Per_B f), \quad B \in \mathcal{B}_0.$$

The remaining induced FT computations can be determined by complex multiplications implied by theorem(E.8). The X-invariance of f reduces the number of required induced FT computations.

For any subgroup $B < A$, define

$$X_B = \{x \in X : \alpha_x(B) = B\}.$$

X_B is a subgroup of X and acts on $L(A/B)$.

Theorem E.10 *If f is X-invariant then $Per_B f \in L(A/B)$ is X_B-invariant.*

By the theorem the induced FT computations

$$F_{\phi_1}^B(Per_B f), \quad B \in \mathcal{B}_0$$

are taken on X_B-invariant data. To make full use of the X-invariance of f we must provide code which make full use of the X_B-invariance of $Per_B f$, $B \in \mathcal{B}_0$. In 1 or 2-D, affine group invariant FFT algorithms are substantially simpler due to the restricted class of 1 or 2-D affine group actions (see appendix).

X-invariant RT algorithm Choose an $\dot{X}$-invariant dual covering $\mathcal{B}$ of A and a complete system $\mathcal{B}_0$ of $\dot{X}$-orbit representatives in $\mathcal{B}$.

- Form the periodizations

$$Per_B f \in L(A/B), \quad B \in \mathcal{B}_0.$$

- Compute X_B-invariant FT

$$F^B_{\phi_1}(Per_B f), \quad B \in \mathcal{B}_0.$$

- Compute

$$F^B_{\phi_1}(Per_B f), \quad B \in \mathcal{B},$$

by Eq. (60).

Example E.17 *Affine group-invariant RT*

There are 5 affine crystallographic groups whose point group is $P6$.

Table E.8 *Affine groups with point group P6*

group	generator
$P6_1$	$(0,0,M,\alpha)$
$P6_2$	$(0,0,2M,\alpha)$
$P6_3$	$(0,0,3M,\alpha)$
$P6_4$	$(0,0,4M,\alpha)$
$P6_5$	$(0,0,5M,\alpha)$

RT algorithm proceeds as in the case of $P6$. Now the invariance condition on FT is given by Eq.(60). For $0 \leq l \leq 5$, a $P6_l$-invariant $f \in L(A)$, the induced FT of the $D_{j,k}$-periodization of f determines $\hat{f}$ on $D_{j,k}^{\perp} \in \mathcal{B}_0$. To determine $\hat{f}$ on $P6^{\#}$-orbits of $D_{j,k}^{\perp}$ set

$$< (c_1,c_2,c_3), \phi(0,0,M) >= w = exp^{\frac{-2\pi i}{6}}.$$

$$\begin{aligned}
\hat{f}(c_1,c_2,c_3) &= w^{c_3 l}\hat{f}(\alpha^{\#}(c_1,c_2,c_3)) \\
&= w^{2c_3 l}\hat{f}((\alpha^2)^{\#}(c_1,c_2,c_3)) \\
&= w^{3c_3 l}\hat{f}((\alpha^3)^{\#}(c_1,c_2,c_3)) \\
&= w^{4c_3 l}\hat{f}((\alpha^4)^{\#}(c_1,c_2,c_3)) \\
&= w^{5c_3 l}\hat{f}((\alpha^5)^{\#}(c_1,c_2,c_3)), \quad 1 \leq l \leq 5.
\end{aligned}$$

$1 \leq l \leq 5$.

The group that contains all of the 48 tetragonal crystallographic groups is $P4/mmm$. As in the case of $P6/mmm$, once a $P4/mmm^{\#}$-invariant covering subgroup is partitioned into $P4/mmm^{\#}$-orbits, a code for the RT algorithm with respect to this partitioning contains codes for FT computation of functions invariant under subgroups of $P4/mmm$.

One can also choose a group that contains all the crystallographic point groups; This group need not a crystallographic group.

E.6.4 $X^{\#}$-invariant RT algorithm

Consider a subgroup X of $Aff(A)$. In many applications we will have to compute the inverse FT of $X^{\#}$-invariant data. Up to index reversal, this problem is equivalent to computing the

FT of $X^{\#}$-invariant data. We will embed this problem in the second form RT algorithm. In problems requiring several stages of FT and inverse FT, it makes sense to follow the first form RT algorithm which outputs decimated data by the second form RT algorithms which inputs decimated data and conversely, removing the necessity of data rearrangement steps at each cycle.

In the second form of RT algorithm we compute $F_\phi f$, $f \in L(A)$ by first computing the collection of induced FT

$$F^B_{\phi_2}(Dec^\mu_B f), \quad B \in \mathcal{B}.$$

Theorem E.11 *For a subgroup $B < A$, if $f \in L(A)$ is $X^{\#}$-invariant, then*

$$F_\phi(Dec_B f)(-a) = F_\phi(Dec_{\alpha^{\#}_x B} f)(-xa), \quad a \in A,\ x \in X. \tag{61}$$

Proof

$$\begin{aligned} F_\phi(Dec_B f)(-c) &= \sum_{b \in B} f(b) < b, \phi(c) > \\ &= \sum_{b \in B} f(\alpha^{\#} b) < b, \phi(c - \alpha_x^{-1} a_x) > \\ &= \sum_{b \in \alpha^{\#}_x B} f(b) < b, \phi(\alpha_x c - a_x) > \\ &= F_\phi(Dec_{\alpha^{\#}_x B} f(-xc). \end{aligned}$$

Choose an $\dot{X}^{\#}$-invariant covering $\mathcal{B}$ of A and a complete system $\mathcal{B}_0$ of $\dot{X}^{\#}$-orbit representatives in $\mathcal{B}$. It suffices to compute the collection of induced FT

$$F^B_{\phi_2}(Dec_B \tilde{f}), B \in \mathcal{B}_0$$

The remaining induced FT computations can be computed from the theorem.

Set

$$X_B = \{x \in X : \alpha_x(B) = B\}.$$

Theorem E.12 *For $X^{\#}$-invariant $f \in L(A)$ and $B < A$, $Dec_B f$ is $\dot{X}^{\#}$-invariant.*

$$Dec_B f(b) = \overline{< a_x, \phi(\alpha^{\#}_x b) >} Dec_B f(\alpha^{\#}_x b), \quad b \in B,\ x \in X_B.$$

In 3D crystallographic applications, specialized routines as described in the preceding two subsections can be applied to these induced FT computations.

E.7 Implementation Results

We have implemented symmetrized 3D crystallographic FFTs for the case of $P6$ symmetric data. The data is assumed to be defined on the $\mathbf{Z}/3N \times \mathbf{Z}/3N \times \mathbf{Z}/6M$ lattice, where N and M are powers of two.

Algorithm 1

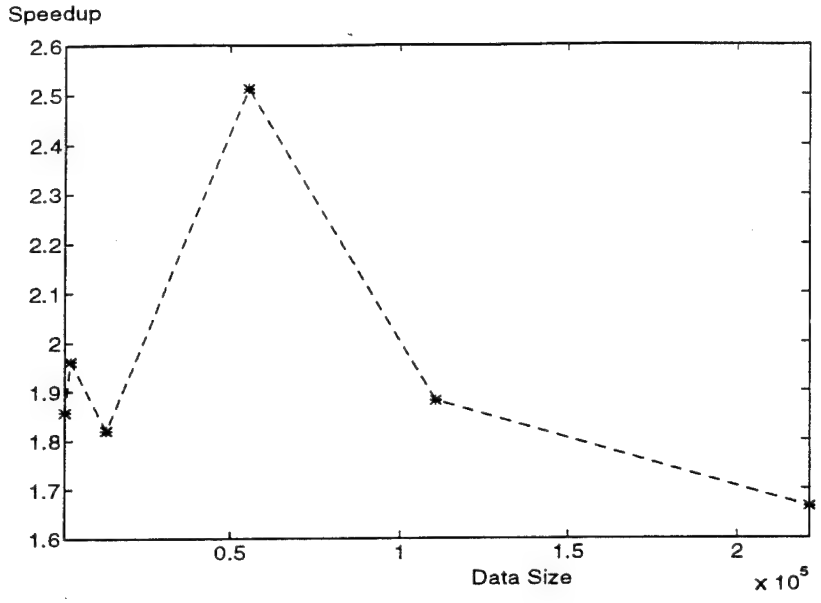
1. Use CRT to re-index the data set such that the problem is transformed to an equivalent 5D computation:
   $\mathbf{Z}/3N \times \mathbf{Z}/3N \times \mathbf{Z}/6M \longrightarrow \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/N \times \mathbf{Z}/N \times \mathbf{Z}/6M$.
   Although this step is computationally expensive, involving irregular accessing of the data stored in the main memory, it should be noted that in many applications where a large number of iterations of the forward and inverse FFT are required, the CRT re-indexing can be carried out only once and then the optimization can be performed in the 5D domain.

2. Apply the RT algorithm to the $\mathbf{Z}/3 \times \mathbf{Z}/3$ to compute the periodized data on two out of the total four subgroups. The periodization results in two distinct data sets, A_1 and A_2, each defined on $\mathbf{Z}/3 \times \mathbf{Z}/N \times \mathbf{Z}/N \times \mathbf{Z}/6M$.

3. Perform two 4D FFTs on the data sets A_1 and A_2 to implement the induced FT. The sets A_1 and A_2 are $P2$ and $P6$ symmetric correspondingly, such that efficient symmetrized FFT code can be used for the computations.

If symmetrized FFT code is not used in step 3, the computational savings are roughly in the order of 1/2. In Figure E.1 we plot the speedup over the non-symmetrized FFT versus the size of the data set.

Figure E.1 *Speedup of the P6 symmetrized FFT over the non-symmetrized FFT versus the data size. Symmetrized RTA on* $\mathbf{Z}/3 \times \mathbf{Z}/3$.



The second implementation results in even more speedups over the non-symmetrized FFT:

Algorithm 2

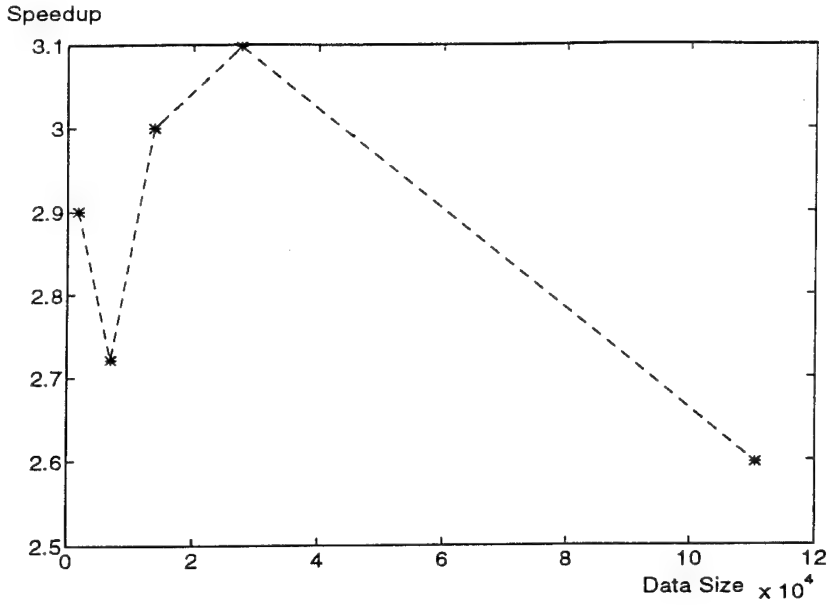
1. Use the CRT to re-index the data set such that the problem is transformed to an equivalent 5D computation:
   $\mathbf{Z}/3N \times \mathbf{Z}/3N \times \mathbf{Z}/6M \longrightarrow \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/N \times \mathbf{Z}/N \times \mathbf{Z}/6M$.

2. Apply the RT algorithm on $\mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/N \times \mathbf{Z}/N$ and compute the periodized data on one third of the total $4 \times (3/2)N$ subgroups. The periodization results in $2N$ distinct data sets, each defined on $\mathbf{Z}/6M$.

3. Perform $2N$ independent 1D FFTs on data of length $6M$. These distinct data sets are $P2$ symmetric, so that efficient $P2$-symmetrized FFT code can be used.

If symmetrized FFT code is not used in step 3, the computational savings are roughly in the order of 1/3. In Figure E.2 we plot the speedup over the non-symmetrized FFT versus the size of the data set. If $P2$ symmetrized FFT code is used, the computational savings are roughly in the order of 1/6 which is the theoretical maximum since the original data are $P6$ symmetric.

Figure E.2 *Speedup of the P6 symmetrized FFT over the non-symmetrized FFT versus the data size.*



The P6 symmetrized RT algorithm based FFTs share the highly parallelizable structure of the general RT algorithm. A variety of choices of a multiprocessor algorithm are available allowing for efficient implementations depending on the characteristics of the particular platform. Consider for example Algorithm 1. If two processors are available and all of the $2 \cdot 3 \cdot N \cdot N \cdot 6M$ data set is stored in each processor, no-interprocessor communication is needed since each processor can independently compute the periodization and 4D FFT. If only half of the data is stored in the memory of each processor, then in order to compute the periodizations, each processor has to send its data to the other, resulting in a total amount of communication (number of processors $\times$ size of messages) equal $2 \cdot 3 \cdot N \cdot N \cdot 6M$.

If $P > 2$ processors are available, the data can be divided along the last dimension into sets of size $2 \cdot 3 \cdot N \cdot N \cdot 6M/P$, each set being stored into the local memory of one processor. After the computation of the periodizations, each processor keeps $3 \cdot N \cdot N \cdot 6M/P$ of local data, and then performs local FFTs along the first three dimensions. To complete the computation, FFTs along the last dimension have to be performed. Since the data are distributed among the processors along the last dimension, a global transposition is required: Each processor keeps $1/P$ of its local data, and sends $(P-1)/P$ data to other processors. The total communication requirements are then: $(P-1)\times$ local data size $= (P-1) \times 3 \cdot N \cdot N \cdot 6M/P$. In an alternative implementation, P processors are being divided into $P/2$ clusters of two processors, with local data being duplicated within each cluster. In this implementation, each node stores twice as many data as before, but the efficiency can be increased in certain multiprocessor networks

since now the global transposition step is replaced with two independent global transpositions each involving only $P/2$ nodes.

E.7.1 Complexity

E.7.2 Row-Column Algorithm

Set

$$A = \mathbf{Z}/3N \times \mathbf{Z}/3N \times \mathbf{Z}/3M.$$

The computation of the 3D FT using conventional row-column algorithm of processing the data dimension at a time on many parallel systems pays considerably higher price on interprocessor communication than FT computation. RT algorithm offers an alternate data movements in MD FT computation. We list some performance results here.

GT/RT algorithm I

Using CRT,

$$A \simeq A_1 \times A_2 = (\mathbf{Z}/3 \times \mathbf{Z}/3) \times (\mathbf{Z}/3 \times \mathbf{Z}/N \times \mathbf{Z}/N \times \mathbf{Z}/M).$$

Data reduction (periodization) stage costs $4 \times 2 \times 3N^2M$ additions, which can be combined with data loading operation in a broadcasting mode; on some parallel systems it is given for free. In a 4 processor system, each processor carries out $2 \times 3N^2M$ additions, while receiving input data, followed by a local 5D $3 \times 3 \times N \times N \times M$. FT computation. This algorithm eliminates interprocessor communication completely, and each processor has balanced load with uniform computation format.

E.7.3 GT/RT algorithm II

$$A \simeq A_1 \times A_2 = (\mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3) \times (\mathbf{Z}/N \times \mathbf{Z}/N \times \mathbf{Z}/M).$$

In this decomposition, each processor carries out $(2 \times 3) \times N^2M$ additions to implement periodization while receiving input data, followed by a local 4D $3 \times N \times N \times M$ FT computation. This decomposition is well suited on a 13 processor system. Both reduction and FT computation are carried out in parallel.

The RT algorithms I and II show uniform decomposition of a 3D problem into subsets. The combination of RT algorithms with other fast algorithms will provide a highly scalable feature that can be matched to various degrees of parallelism and granularity of a parallel system.

The RT algorithm partitions input data at the global level to match each subset into node processors, carrying out loading and reduction operations concurrently at each node, then FT computations are performed in parallel.

In tables E.9, E.10, timing results on the Intel iPSC/860 with 4 and 8 node implementations are given. The timing results of the next power of 2 sizes of Intel FFT library are also included for comparison. (Non-power of 2 routines are not available in the standard library.) The GT/RT algorithm I was implemented on the 4-node hypercube architecture.

The periodization (reduction stage) is coded in standard Fortran whereas the FFT and 3-point FT calls on the Kuck & Associates optimized assembly routines and our own vectorized 3-point FT routines respectively.

Table E.9 *Timing Results on iPSC/860 (3-D) (4-nodes)*

GT/RT (4-nodes)		Row-Column (4-nodes)	
size	time	size	time
$48 \times 48 \times 48$	360 ms	$64 \times 64 \times 64$	566 ms
$48 \times 48 \times 96$	572 ms	$64 \times 64 \times 128$	1122 ms
$48 \times 96 \times 96$	980 ms	$64 \times 128 \times 128$	2202 ms

Table E.10 *Timing Results on iPSC/860 (3-D) (4-nodes)*

GT/RT (4-nodes)		Row-Column (8-nodes)	
size	time	size	time
$48 \times 48 \times 48$	360 ms	$64 \times 64 \times 64$	282 ms
$48 \times 48 \times 96$	572 ms	$64 \times 64 \times 128$	585 ms
$48 \times 96 \times 96$	980 ms	$64 \times 128 \times 128$	1152 ms
$96 \times 96 \times 96$	2029 ms	$128 \times 128 \times 128$	2276 ms

E.8 Affine group CT FFT

The global decomposition stage of a CT FFT algorithm computes pseudo-periodizations relative to a subgroup B of the indexing group A. In this chapter we present a CT FFT algorithm whose pseudo-periodizations are taken relative to an abelian subgroup $X < Aff(A)$. In the classical case, X consists of pure translations. If Y is a subgroup of X the CT FFT algorithm associated to X can easily be adopted to produce an FFT algorithm for Y-invariant data. Code

which implements this CT FFT produces by a process of disabling, Y-invariant FFT code for every subgroup Y of X.

For applications, the choice of X is motivated by two factors. First code for the CT FFT associated to X should be simple to write, scalable and efficient. Second X should contain a large collection of subgroups of interest in applications.

E.8.1 Extended CT FFT: abelian point group

Choose $f \in L(A)$ and an abelian subgroup G of $Aut(A)$. For $\gamma^* \in G^*$ define the *pseudo-periodizations* $f_{\gamma^*} \in L(A)$ by

$$f_{\gamma^*}(a) = \sum_{\gamma \in G} f(\gamma) < \gamma, \gamma^* >, \quad a \in A. \tag{62}$$

Since

$$\sum_{\gamma^* \in G^*} < \gamma, \gamma^* > = \begin{cases} o(G), & \gamma = \text{identity map}, \\ 0, & \text{otherwise}, \end{cases} \tag{63}$$

we can write

$$f = \frac{1}{o(G)} \sum_{\gamma^* \in G^*} f_{\gamma^*}. \tag{64}$$

We can compute $F_\phi f$ by computing the collection of FT's

$$F_\phi f_{\gamma^*}, \quad \gamma^* \in G^*. \tag{65}$$

We have replaced a single FT computation by a collection of FT computations. However, the pseudo-periodizations satisfy the following group invariance property.

Theorem E.13 *For $\gamma^* \in G^*$,*

$$f_{\gamma^*}(\gamma(a)) = \overline{< \gamma, \gamma^* >} f_{\gamma^*}(a), \quad a \in A, \gamma \in G.$$

$$F_\phi f_{\gamma^*}(\gamma^\#(a)) = < \gamma, \gamma^* > F_\phi f_{\gamma^*}(a), \quad a \in A, \gamma \in G.$$

We will say that f_{γ^*} is G-invariant with character . The CT FFT associated to G decomposes the computation of $F_\phi f$ into a collection of FT computations on G-invariant with character data which can be implemented by simple modifications of the point group RT algorithm.

Suppose K is a subgroup of G. If we begin with a K-invariant data, we can reduce the number of FT computations. Set

$$K_* = \{\gamma^* \in G^* : < \kappa, \gamma^* > = 1, \text{ for all } \kappa \in K\}. \tag{66}$$

K_* is a subgroup of G^* isomorphic to the character group $(G/K)^*$. Choose a complete set of representatives of K-cosets in G

$$\gamma_0, \gamma_1, \cdots, \gamma_{L-1}. \tag{67}$$

Then every $g \in G$ can be written uniquely in the form

$$\gamma = \kappa\gamma_l, \quad \kappa \in K, 0 \leq l < L. \tag{68}$$

Theorem E.14 *If $f \in L(A)$ is K-invariant then the pseudo-periodization f_{γ^*} vanishes unless $\gamma^* \in K_*$.*

$$\begin{aligned} \textbf{Proof} \quad f_{\gamma^*}(a) &= \sum_{l=0}^{L-1} \sum_{\kappa \in K} f(\kappa\gamma_l a) < \kappa\gamma_l, \gamma^* > \\ &= \sum_{l=0}^{L-1} f(\gamma_l a) < \gamma_l, \gamma^* > \sum_{\kappa \in K} < \kappa, \gamma^* > \end{aligned}$$

by K-invariance. Since $\sum_{\kappa \in K} < \kappa, \gamma^* >$ vanishes unless $\gamma^* \in K_*$, the proof of the theorem is complete.

Code for the CT FFT algorithm associated to G applies to the computation of the FT of the K-invariant data, $K < G$, by disabling all the pseudo-periodizations corresponding to $\gamma^* \notin K_*$.

E.8.2 CT FFT with respect to *Pmmm*

For $\rho, \tau \in Pmmm$,

$$\rho = \rho_1^{r_1}\rho_2^{r_2}\rho_3^{r_3}, \quad \tau = \rho_1^{t_1}\rho_2^{t_2}\rho_3^{t_3},$$

define

$$< \rho, \tau^* >= (-1)^{r_1t_1 + r_2t_2 + r_3t_3}.$$

Associate with the function $f \in L(A)$, the column vector $\mathbf{f}_0$ of length $K = 8NML$ by listing $f(a_1, a_2, a_3)$, antilexicographic ordering of $(a_1, a_2, a_3) \in A$. Also define the vectors $\mathbf{f}_{8j}$, $0 \leq j \leq 7$ by listing $f(s_{8j}(a_1, a_2, a_3))$, in order of $(a_1, a_2, a_3) \in A$. The the generalized periodizations of f

with respect to $Pmmm$ can be implemented by the vector additions

$$\begin{bmatrix} \mathbf{f}_{s_0^*} \\ \mathbf{f}_{s_8^*} \\ \mathbf{f}_{s_{16}^*} \\ \mathbf{f}_{s_{24}^*} \\ \mathbf{f}_{s_{32}^*} \\ \mathbf{f}_{s_{40}^*} \\ \mathbf{f}_{s_{48}^*} \\ \mathbf{f}_{s_{56}^*} \end{bmatrix} = [F(2) \otimes I_K \otimes F(2) \otimes F(2) \otimes I_K] \begin{bmatrix} \mathbf{f}_{s_0} \\ \mathbf{f}_{s_8} \\ \mathbf{f}_{s_{16}} \\ \mathbf{f}_{s_{24}} \\ \mathbf{f}_{s_{32}} \\ \mathbf{f}_{s_{40}} \\ \mathbf{f}_{s_{48}} \\ \mathbf{f}_{s_{56}} \end{bmatrix}, \tag{69}$$

where $F(2)$ denotes the 2-point FT matrix,

$$F(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and I_K is the $K \times K$ identity matrix.

Crystallographic group $P2$ [13] is a subgroup of $Pmmm$,

$$P2 = \{1, s_{24}\}.$$

$$P2_* = \{1, s_{24}, s_{32}, s_{56}\}.$$

If $f \in L(A)$ is $P2$-invariant, then 4 of the periodizations vanish. Each of the non-vanishing periodizations are $Pmmm$-invariant up to multiplication by ± 1, and FT is computed with this invariance.

Another crystallographic subgroup of $Pmmm$ is $P222$.

$$P222 = \{1, s_{24}, s_{40}, s_{48}\}.$$

$$P222_* = \{1, s_{56}\}.$$

For $P222$-invariant f, all the periodizations except $f_{s_0^*}$ and $f_{s_{56}^*}$ vanish.

If f is $Pmmm$-invariant, then computation is carried out only for $f_{s_0^*}$.

E.8.3 Extended CT FFT : abelian affine group

The discussion of section E.8.1 will be extended to abelian subgroups X of $Aff(A)$ of the form $X = B \times K$ where B is a subgroup of A and K is a subgroup of $Aut(A)$. The CT FFT algorithm associated to X combines features of the standard CT FFT associated to B and the

abelian point group CT FFT associated to K. The pseudo-periodizations are now taken with respect to the affine subgroup X. The motivation is to unify the writing of FT code for affine group invariant data.

Choose an abelian subgroup X of $Aff(A)$ of the form $X = B \times K$. Then $X^* = B^* \times K^*$. We will usually write bk for (b, k) and b^*k^* for (b^*, k^*). Denote a complete set of $B^\perp$-coset representatives by

$$z(b^*) = \phi_2^{-1}(b^*), \quad b^* \in B^*. \tag{70}$$

For $f \in L(A)$, define the *pseudo-periodizations* $f_{x^*} \in L(A)$, $x^* \in X^*$, by

$$f_{x^*}(a) = \sum_{x \in X} f(xa) < x, x^* >, \quad a \in A,\ x^* \in X^*. \tag{71}$$

$$f_{x^*}(x(a)) = \overline{< x, x^* >} f_{x^*}(a), \quad a \in A,\ x^* \in X^*. \tag{72}$$

Since

$$f = \frac{1}{o(X)} \sum_{x^* \in X^*} f_{x^*}. \tag{73}$$

we can compute $F_\phi f$ by the collection of FT computations

$$F_\phi f_{x^*}, \quad x^* \in X^*.$$

A direct computation shows that f_{x^*} satisfies the group invariance with character condition. In particular

$$f_{x^*}(b + a) = \overline{< b, b^* >} f_{x^*}(a), \quad b \in B,\ x^* = b^*k^* \in X^*. \tag{74}$$

Define $g_{x^*} \in L(A)$, $x^* \in X^*$, by

$$g_{x^*}(a) = f_{x^*}(a) < a, \phi(z(b^*)) >, \quad a \in A,\ \ x^* = b^*k^*. \tag{75}$$

g_{x^*} is B-invariant and can be viewed as a function in $L(A/B)$.

Theorem E.15 *For $x^* = b^*k^* \in X^*$, $F_\phi f_{x^*}$ vanishes off of $z(b^*) + B^\perp$ and we have*

$$F_\phi f_{x^*}(z(b^*) + b^\perp) = o(B) F_{\phi_1} g_{x^*}(b^\perp), \quad b^\perp \in B^\perp.$$

Proof Choose a complete system of representatives for the B-cosets in A

$$m_1, \cdots, m_J.$$

Setting

$$c = z(b_1^*) + b^\perp, \quad b_1^* \in B^*, b^\perp \in B^\perp$$
$$a = m_j + b, \quad 1 \le j \le J, b \in B,$$

in

$$F_\phi f(c) = \sum_{a \in A} f(a) \ < a, \phi(c) >$$

we have, applying Eq. (74)

$$F_\phi f(c) = \sum_{j=1}^{J} f_{x^*}(m_j) \ < m_j, \phi(z(b_1^*) + b^\perp) > \sum_{b \in B} < b, b_1^* - b^* >$$

which vanishes unless $b_1^* = b^*$, proving $F_\phi f$ vanishes off of $z(b^*) + B^\perp$. Then by theorem E.6,

$$F_\phi f(z(b^*) + b^\perp) = o(B) \sum_{j=1}^{J} g_{x^*}(m_j) \ < m_j, \phi(b^\perp) >,$$

completing the proof of the theorem.

For $b^* \in B^*$ define

$$S(b^*) = \{g_{b^*k^*} : k^* \in K^*\}. \tag{76}$$

By theorem E.15

$$F_\phi f(z(b^*) + b^\perp) = \frac{o(B)}{o(X)} \sum_{k^* \in K^*} F_{\phi_1} g_{b^*k^*}(b^\perp), \quad b^\perp \in B^\perp, \tag{77}$$

which implies that $F_\phi f$ on the coset

$$z(b^*) + B^\perp, \ \ b^* \in B^*,$$

is determined by the induced FT of functions in $S(b^*)$.

The pseudo-periodization operations introduce data redundancies which we will now describe.

Set $C = A/B$. K acts by the identity mapping on B and induces a group of automorphisms of C denoted also by K. For $b^* \in B^*$ and $k \in K$, there exists a unique $\zeta_{b^*}(k) \in B^\perp$ such that

$$k^\#(z(b^*)) = z(b^*) + \zeta_{b^*}(k). \tag{78}$$

Direct computation shows that

$$k^\#(\zeta_{b^*}(k')) + \zeta_{b^*}(k) = \zeta_{b^*}(kk'), \quad k, k' \in K. \tag{79}$$

Define

$$\zeta_{b^*}^*(k) = \phi_1(\zeta_{b^*}(k)) \in C^*. \tag{80}$$

Theorem E.16 *For $x^* = b^*\kappa^* \in X^*$ and $\kappa \in K$,*

$$g_{x^*}(\kappa(c)) = \overline{<\kappa, \kappa^*>} < c, \zeta^*_{b^*}(\kappa) > g_{x^*}(c), \quad c \in C. \tag{81}$$

$$F_{\phi_1} g_{x^*}(\kappa^{\#}(b^{\perp}) + \zeta_{b^*}(k)) = <\kappa, \kappa^*> F_{\phi_1} g_{x^*}(b^{\perp}), \quad b^{\perp} \in B^{\perp}, \kappa \in K. \tag{82}$$

Proof By Eqs. (72), (75) and (78)

$$\begin{aligned} g_{x^*}(\kappa(a) &= \overline{<\kappa, \kappa^*>} < \kappa(a), \phi(z(b^*)) > f_{x^*}(a), \quad a \in A, \kappa \in K \\ &= \overline{<\kappa, \kappa^*>} < a, \phi(\kappa^{\#}(z(b^*))) > f_{x^*}(a) \\ &= \overline{<\kappa, \kappa^*>} < a, \phi(\zeta_{b^*}(\kappa)) > g_{x^*}(a). \end{aligned}$$

The second statement can be proved by usual arguments. A modified RT algorithm can be applied to the induced FT computations.

For a subgroup Y of X, set

$$Y_* = \{x^* \in X^* :< y, x^* >= 1, \text{for all } y \in Y\}. \tag{83}$$

Arguing as in theorem E.14, we have the following theorem.

Theorem E.17 *If X is a subgroup of $Aff(A)$ and Y is a subgroup of X, then for Y-invariant $f \in L(A)$, the pseudo-periodizations f_{x^*}, $x^* \in X^*$ vanishes unless $x^* \in Y_*$.*

Affine group CT FFT code for X can be used to compute the FT of Y-invariant data, for any subgroup Y of X. In several important applications, the group X can be chosen such that the corresponding CT FFT algorithm can be implemented by simple 1-D routines while more complicated code is required for a direct implementation of the FT of Y-invariant data, Y.

E.8.4 CT FFT with respect to *Fmmm*

We will continue with the notations established in example E.4.

$$Fmmm = B \times Pmmm.$$

We will use the B-periodization computation of example E.11 as the first stage of the two stage pseudo-periodizations with respect to $Fmmm$. Recall the ordering of the elements of $Fmmm$ given in example E.4 :

$$B = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}.$$

$$Pmmm = \{s_0, s_8, s_{16}, s_{24}, s_{32}, s_{40}, s_{48}, s_{56}\}.$$

$$Fmmm = \{s_{8l+k} : 0 \le k, l \le 7\}.$$

For $(a_1 a_2 a_3) \in A$, observe that

$$s_{8l}(a_1, a_2, a_3) = s_{8l+l}(a_1, a_2, a_3) + s_l, \quad s_l \in B.$$

In example E.11, periodizations

$$f_{b_l^*}, \quad 0 \le l \le 7$$

are made on the collection of B-coset representatives

$$C = \{(a_1, a_2, a_3) : 0 \le a_i \le N_i, \ \ i = 1, 2, 3\}.$$

$$\begin{aligned} f_{s^*_{8l+k}}(a) &= \sum_{n=0}^{7}\sum_{m=0}^{7} f(s_{8n}a + s_m) < s_m, s_k^* >< s_{8n}, s_{8l}^* > \\ &= \sum_{n=0}^{7} f_{b_k^*} f(s_{8n}a) < s_{8n}, s_{8l}^* > \\ &= \sum_{n=0}^{7} f_{b_k^*} f(s_{8n+n}a) < s_n, b_k^* >< s_{8n}, s_{8l}^* > \end{aligned}$$

CT FFT with respect to Fmmm was implemented on a Sun4 station [1].

E.9 Incorporating 1D symmetries in FFT

We have developed various FFT algorithms incorporating certain 1D symmetry. In this section, we give an example of incorporating invariance conditions in data without giving up the use of highly efficient FFT routines.

Set $A = \mathbf{Z}/N$, for a natural number N. For $f \in L(A)$, the invariance conditions we will consider here are

$$f(a) = \pm f(-a). \tag{84}$$

An efficient algorithm was given by Cooley et al. [10] and Rabiner [16] which reduced the computation to that for an $N/2$-point FFT with preprocessing and postprocessing. The procedures are summarized as follows.

a. Compute

$$V(0) = 2 \sum_{a=0}^{N/4-1} f(2a+1).$$

b. For $a = 1, 2, ..., N/4 - 1$, formulate the sequence $g(a)$ as

$$\begin{aligned} g(a) &= f(2a) + [f(2a+1) - f(2a-1)], \\ g(N/2-a) &= f(2a) - [f(2a+1) - f(2a-1)], \\ g(0) &= f(0), \\ g(N/4) &= f(N/2). \end{aligned}$$

c. Take the $N/2$-point FFT of $g(a)$; call this result $G(b)$.

d. Form two sequences

$$\begin{aligned} U(b) &= Re[G(b)], \quad b = 0, 1, 2, ..., N/4, \\ V(b) &= \frac{Im[G(b)]}{2sin(2\pi b/N)}, \quad b = 1, 2, ..., N/4 - 1. \end{aligned}$$

e. For $b = 1, 2, ..., N/4$, the transformed data sequence $F(b)$ is given as

$$\begin{aligned} F(b) &= U(b) + V(b), \\ F(N/2-b) &= U(b) - V(b), \\ F(0) &= U(0) + V(0), \\ F(N/2) &= U(0) - V(0). \end{aligned}$$

Notice that in step **d**, the computation involves division by $\{sin(2\pi b/N)\}$. This may cause stability problem for large size N.

We summarize here an algorithm proposed in [15], to overcome the stability problem.

a. Form two sequences

$$\begin{aligned} h(a) &= f(a) + f(N/2-a), \quad a = 0, 1, 2, ..., N/4, \\ g(a) &= [f(a) - f(N/2-a)]cos(2\pi a/N), \quad a = 0, 1, 2, ..., N/4, \end{aligned}$$

both $h(a)$ and $g(a)$ have invariance conditions.

b. Take the $N/2$-point(half size) symmetric FT of $h(a)$ and $g(a)$.

c. The transformed data sequence $F(b)$ is given as

$$\begin{aligned} F(2b) &= H(b), \quad b = 0, 1, 2, ..., N/4 - 1, \\ F(1) &= G(0), \\ F(2b+1) &= 2G(b) - F(2b-1), \quad b = 1, 2, ..., N/4 - 1. \end{aligned}$$

This algorithm can be recursively used for transform size of $N = 2^m$ or $N = 2^m l$, where $m > 1$ and l an odd number.

In step **a**, multiplications by $\{cos(2\pi a/N)\}$ are required to formulate $g(a)$. If, however, N is twice an odd number, then an alternative procedure, based on the Good-Thomas prime factor algorithm [12, 18], can be used to avoid these multiplications. In this case, the computational procedures can be stated as

a. Take the $N/2$-point (half size) symmetric FFT of $f_1(a) = f(2a)$ and $f_2(a) = f(N/2+2a)$; call them $F_1(b)$ and $F_2(b)$ respectively.

b. For $b = 0, 1, 2, ..., (N/2-1)/2$, the transformed data sequence $F(b)$ is given as

$$\begin{aligned} F(2b) &= F(N-2b) = F_1(2b) + F_2(2b), \\ F(N/2+2b) &= F(N/2-2b) = F_1(2b) - F_2(2b). \end{aligned}$$

If the data is real, the same algorithm can be used with half size real FFTs. The saving in FFT computation will be approximately 50 percent in comparison with complex data.

References

[1] Y. Abdelatif, *Periodization and Decimation for FFTs and crystallographic FFTs*, Ph.D Thesis, CCNY, CUNY, 1994.

[2] M. An, I. Gertner, M. Rofheart and R. Tolimieri, "Discrete Fast Fourier Transform Algorithms: A Tutorial Survey," *Advances in Electronics and Electron Physics*, **80** Academic Press, 1991.

[3] M. An, J. W. Cooley and R. Tolimieri, "Factorization Method for Crystallographic Fourier Transforms", *Advances in Applied Mathematics*, **11**, 358-371, (1990).

[4] M. An, C. Lu, E. Prince and R. Tolimieri, "Fast Fourier Transform Algorithms for Real and Symmetric Data", *ACTA Cryst.*, (1992). A48, 415-418.

[5] M. An, C. Lu, E. Prince and R. Tolimieri, "Fast Fourier Transforms for Space Groups Containing Rotation Axes of Order Three and Higher", *ACTA Cryst.*, (1992). A48, 346-349.

[6] N. Anupindi and K. M. Prabhu, "Split-Radix FHT Algorithm for Real-Symmetric Data", *Electronics Letters*, 8th Nov. 90, Vol. 26, No. 23, 1973-1975.

[7] G. Bricogne, "Geometric Sources of Redundency in Intensity Data and their Use for Phase Determination", *Acta Cryst.*, **A30**, (1974), 395-405.

[8] G. Bricogne and R. Tolimieri, "Symmetrized FFT Algorithms", *The IMA Volumes in Mathematics and Its Applications*, vol. 23, Springer-Verlag, New York/Berlin, 1990.

[9] C. S. Burrus, "Index Mappings for Multidimensional Formulation of the DFT and Convolution", *IEEE Trans. on ASSP*, vol. ASSP-25, 239-242, June 1977.

[10] J. W. Cooley, P. A. Lewis and P. D. Welch, "The Fast Fourier Transform Algorithms: Programming Considerations in the Calculation of Sine, Cosine and Laplace Transforms", *J. Sound Vib.*, Vol. 12, 315-337, July 1970.

[11] I. Gertner, "A new efficient algorithm to compute the twodimensional discrete Fourier transform," *IEEE Trans. ASSP* **37**(7), 1036-1050, 1988.

[12] I. J. Good, "The Interaction Algorithm and Practical Fourier Analysis", *J. R. Statis. Soc. B.*, vol. 20, No. 2, 1958.

[13] N.F.M. Henry and K. Londsdale, Ed., *International Tables for X-Ray Crystallography*, Volume I, 1952, The Kynoch Press, England.

[14] G. Kechriotis, M. An, M.Bletsas, E. Manolakos and R. Tolimieri, "A hybrid approach for computaing multidimensional DFT's on parallel machines and its implementation on the iPSC/860 hypercube," accepted for publication in *IEEE Trans. Signal Proc.*, August, 1993.

[15] C. Lu and R. Tolimieri, "New Algorithms for the FFT Computation of Symmetric and Translational Complex Conjugate Sequences", *the Proceedings of IEEE 1992 International Conference on ASSP*, March 23-26, 1992.

[16] L. Rabiner, "On the Use of Symmetry in FFT Computation", *IEEE Trans. on ASSP*, Vol. ASSP-27, No. 3, June 1979.

[17] L. F. Ten Eyck, "Crystallographic Fast Fourier Transforms", *ACTA Cryst.*, (1973). **A29**, 183-191.

[18] L. H. Thomas, "Using a Computer to Solve Problems in Physics, Application of Digital Computers", *Ginn and Co.* Boston, MAss. 1963.

[19] R. Tolimieri, M. An and C. Lu, *Mathematics of Multidimensional Fourier Transform Algorithms*, Springer-Verlag, New York/Berlin, 1993.

[20] R. Tolimieri, M. An and C. Lu, *Algorithms for Discrete Fourier Transform and Convolutions* Springer-Verlag, New York/Berlin, 1989.